Package ‘misty’

October 14, 2026

Type Package

Title Miscellaneous Functions 'T. Yanagida'

Version 0.8.4

Date 2026-10-14

Author Takuya Yanagida [aut, cre]

Maintainer Takuya Yanagida <takuya.yanagida@univie.ac.at>

Description Miscellaneous functions for (1) data handling (e.g., grand-mean and group-mean centering, coding variables and reverse coding items, scale and cluster scores, reading and writing Excel and SPSS files), (2) descriptive statistics (e.g., frequency table, cross tabulation, effect size measures), (3) missing data (e.g., descriptive statistics for missing data, missing data pattern, Little's test of Missing Completely at Random, and auxiliary variable analysis), (4) multilevel data (e.g., multilevel descriptive statistics, within-group and between-group correlation matrix, multilevel confirmatory factor analysis, level-specific fit indices, cross-level measurement equivalence evaluation, multilevel composite reliability, and multilevel R-squared measures), (5) item analysis (e.g., confirmatory factor analysis, coefficient alpha and omega, between-group and longitudinal measurement equivalence evaluation), (6) statistical analysis (e.g., bootstrap confidence intervals, collinearity and residual diagnostics, dominance analysis, between- and within-subject analysis of variance, latent class analysis, t-test, z-test, sample size determination), and (7) functions to interact with 'Blimp' and 'Mplus'.

Depends R (>= 4.3.0)

License MIT + file LICENSE

Imports data.table, ggplot2, haven, lavaan, lme4, rstudioapi

Suggests boot, clubSandwich, gsl, hdf5r, ic.infer, Matrix, mice, mnormt, mvnmle, mvtnorm, nleqslv, nlme, patchwork, readxl, robustlmm, plyr, writexl

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2026-10-14 10:20:02 UTC

Contents

aov.b . . . 4
aov.w . . . 8
blimp . . . 13
blimp.bayes . . . 17
blimp.plot . . . 22
blimp.print . . . 29
blimp.run . . . 32
blimp.update . . . 34
boot.bs . . . 38
center . . . 42
check.collin . . . 50
check.outlier . . . 53
check.resid . . . 57
chr.color . . . 62
chr.grep . . . 63
chr.gsub . . . 65
chr.omit . . . 66
chr.trim . . . 67
chr.trunc . . . 69
ci.cor . . . 70
ci.mean . . . 80
ci.mean.diff . . . 86
ci.mean.w . . . 89
ci.prop . . . 92
ci.prop.diff . . . 97
ci.var . . . 100
clear . . . 105
cluster.rwg . . . 106
cluster.scores . . . 108
coding . . . 110
coeff.robust . . . 112
coeff.std . . . 118
cohens.d . . . 123
cor.matrix . . . 128
crosstab . . . 132
data.items . . . 135
descript . . . 136
df.check . . . 140
df.duplicated . . . 141
df.head . . . 143
df.long . . . 145
df.merge . . . 148
df.move . . . 150
df.rbind . . . 152
df.rename . . . 153
df.sort . . . 154

df.subset . . . 156
difftest.chibarsq . . . 159
dominance . . . 164
dominance.manual . . . 167
effsize . . . 171
freq . . . 173
indirect . . . 176
item.alpha . . . 180
item.cfa . . . 184
item.dfi . . . 195
item.distract . . . 201
item.invar . . . 204
item.noninvar . . . 215
item.omega . . . 224
item.reverse . . . 227
item.scores . . . 229
item.stats . . . 231
lagged . . . 235
libraries . . . 238
modcomp . . . 239
mplus . . . 246
mplus.bayes . . . 249
mplus.lca . . . 254
mplus.lca.summa . . . 259
mplus.plot . . . 266
mplus.print . . . 276
mplus.run . . . 282
mplus.update . . . 284
multilevel.alpha . . . 288
multilevel.cfa . . . 292
multilevel.cor . . . 303
multilevel.descript . . . 309
multilevel.fit . . . 314
multilevel.icc . . . 316
multilevel.indirect . . . 320
multilevel.invar . . . 323
multilevel.omega . . . 330
multilevel.r2 . . . 336
multilevel.r2.manual . . . 344
na.as . . . 348
na.auxiliary . . . 350
na.coverage . . . 354
na.descript . . . 356
na.indicator . . . 358
na.pattern . . . 359
na.prop . . . 362
na.satcor . . . 363
na.test . . . 365

plot.misty.object . . . 370
print.misty.object . . . 385
read.data . . . 386
read.dta . . . 388
read.mplus . . . 390
read.sav . . . 391
read.xlsx . . . 392
rec . . . 395
restart . . . 397
robust.lmer . . . 398
script.copy . . . 401
script.new . . . 402
script.open . . . 403
setsource . . . 405
sim.lavaan . . . 406
size.mean . . . 408
skewness . . . 410
summa . . . 413
test.bm . . . 418
test.homo . . . 422
test.levene . . . 426
test.linear . . . 429
test.norm . . . 433
test.t . . . 437
test.welch . . . 442
test.z . . . 446
uniq . . . 451
write.data . . . 453
write.dta . . . 455
write.mplus . . . 456
write.result . . . 457
write.sav . . . 458
write.xlsx . . . 461

Index **463**

`aov.b` *Between-Subject Analysis of Variance*

Description

This function performs an one-way between-subject analysis of variance (ANOVA) including Tukey HSD post hoc tests for multiple comparison and provides descriptive statistics, effect size measures, and a plot showing bars representing means for each group and error bars for difference-adjusted confidence intervals.

Usage

```
aov.b(formula, data, hypo = FALSE, descript = FALSE, effsize = FALSE,
      weighted = TRUE, correct = FALSE, posthoc = FALSE, adjust = TRUE,
      conf.level = 0.95, digits = 2, p.digits = 3, as.na = NULL, plot = FALSE,
      geom = c("all", "bar", "point", "ci", "jitter"),
      filename = NULL, width = NA, height = NA, dpi = 600,
      write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`formula`	a formula of the form `y ~ group` where `y` is a numeric variable giving the data values and `group` a numeric variable, character variable or factor with more than two values or factor levels giving the corresponding groups.
`data`	a matrix or data frame containing the variables in the formula `formula`.
`hypo`	logical: if `TRUE` (default), null and alternative hypothesis are shown on the console.
`descript`	logical: if `TRUE` (default), descriptive statistics are shown on the console.
`effsize`	logical: if `TRUE`, effect size measures η^2 and ω^2 for the ANOVA and Cohen's d for the post hoc tests are shown on the console.
`weighted`	logical: if `TRUE` (default), the weighted pooled standard deviation is used to compute Cohen's d.
`correct`	logical: if `TRUE`, correction factor to remove positive bias in small samples for is used to compute Cohen's d.
`posthoc`	logical: if `TRUE`, Tukey HSD post hoc test for multiple comparison is conducted.
`adjust`	logical: if `TRUE` (default), difference-adjustment for the confidence intervals is applied.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`digits`	an integer value indicating the number of decimal places to be used for displaying descriptive statistics and confidence interval.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`plot`	logical: if `TRUE`, a plot showing bar plots with error bars for confidence intervals is drawn. For additional plotting arguments, see `Details` in the help page of the function `plot.misty.object`.
`geom`	a character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"bar"` for drawing bars representing means for each groups, `"point"` for drawing points representing means for each group, `"ci"` for drawing error bars representing confidence intervals, and `"jitter"` for drawing jittered data points.
`filename`	a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"`

(default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument. Note that plots can only be saved when plot = TRUE.

width — a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.

height — a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.

dpi — a numeric value indicating the dpi argument (default is 600) in the ggsave function.

write — a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification is checked.

output — logical: if TRUE (default), output is shown on the console.

... — additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

Confidence Intervals Cumming and Finch (2005) pointed out that when 95% confidence intervals (CI) for two separately plotted means overlap, it is still possible that the CI for the difference would not include zero. Baguley (2012) proposed to adjust the width of the CIs by the factor of $\sqrt{2}$ to reflect the correct width of the CI for a mean difference:

$$\hat{\mu}_j \pm t_{n-1,1-\alpha/2} \frac{\sqrt{2}}{2} \hat{\sigma}_{\hat{\mu}_j}$$

These difference-adjusted CIs around the individual means can be interpreted as if it were a CI for their difference. Note that the width of these intervals is sensitive to differences in the variance and sample size of each sample, i.e., unequal population variances and unequal n alter the interpretation of difference-adjusted CIs.

Value

Returns an object of class misty.object, which is a list with following entries:

call — function call

type — type of analysis

data — data frame with variables used in the current analysis

formula — formula of the current analysis

args — specification of function arguments

plot — ggplot2 object for plotting the results

result — result tables

Author(s)

Takuya Yanagida

References

Baguley, T. S. (2012a). *Serious stats: A guide to advanced statistics for the behavioral sciences*. Palgrave Macmillan.

Cumming, G., and Finch, S. (2005) Inference by eye: Confidence intervals, and how to read pictures of data. *American Psychologist, 60*, 170–80.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

aov.w, test.t, test.z, test.levene, aov.b, cohens.d, ci.mean.diff, ci.mean

Examples

```
#————————————————————————————————————————————————————————————————————
# Between-Subject Analysis of Variance

# Example 1a: Between-Subject ANOVA, default setting
aov.b(hp ~ gear, data = mtcars)

# Example 1b: Print descriptive statistics and Tukey HSD post hoc test
aov.b(hp ~ gear, data = mtcars, descript = TRUE, posthoc = TRUE)

# Example 1c: Print eta-squared and omega-squared
aov.b(hp ~ gear, data = mtcars, effsize = TRUE)

#————————————————————————————————————————————————————————————————————
# Plot

# Example 2a: Plot results, default setting
aov.b(hp ~ gear, data = mtcars, plot = TRUE)

# Example 2b: No bars, draw points representing means and jittered data points
aov.b(hp ~ gear, data = mtcars, plot = TRUE, geom = c("point", "jitter"))

# Example 2c: Plot results, additional arguments
# see Details in the help page of the function plot.misty.object
aov.b(hp ~ gear, data = mtcars, geom = "all", plot = TRUE, jitter.alpha = 0.4,
      title = "Between-Subject ANOVA")

# Example 2d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- aov.b(hp ~ gear, data = mtcars)
plot(object, geom = "jitter", jitter.alpha = 0.4, title = "Between-Subject ANOVA")

#————————————————————————————————————————————————————————————————————
# Create Plot Manually
```

```
# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- aov.b(hp ~ gear, data = mtcars)

# Example 3: Plot
ggplot(object$result$descript, aes(group, y)) +
  geom_bar(aes(group, m), stat = "summary", fun = "mean") +
  geom_jitter(data = object$data, aes(group, y), alpha = 0.1, width = 0.05,
              height = 0, size = 1.25) +
  geom_point(aes(group, m), stat = "identity", size = 3) +
  geom_errorbar(aes(group, m, ymin = low, ymax = upp), width = 0.1) +
  theme_bw()

## Not run:
#-----------------------------------------------------------------------
# Write Results and Save Plot

# Example 4a: Write results into a text file
aov.b(hp ~ gear, data = mtcars, write = "ANOVA.txt")

# Example 4b: Write results into an Excel file
aov.b(hp ~ gear, data = mtcars, write = "ANOVA.xlsx")

# Example 4c: Save plot as PNG fine
aov.b(hp ~ gear, data = mtcars, plot = TRUE, filename = "ANOVA.png",
      width = 6, height = 5)

## End(Not run)
```

---

aov.w *Repeated Measures Analysis of Variance (Within-Subject ANOVA)*

Description

This function performs an one-way repeated measures analysis of variance (within subject ANOVA) including paired-samples t-tests for multiple comparison and provides descriptive statistics, effect size measures, and a plot showing error bars for difference-adjusted Cousineau-Morey within-subject confidence intervals with jittered data points including subject-specific lines.

Usage

```
aov.w(formula, data, print = c("all", "none", "GG", "HF", "LB"), hypo = FALSE,
      epsilon = FALSE, descript = FALSE, effsize = FALSE, posthoc = FALSE,
      na.omit = TRUE, adjust = TRUE, conf.level = 0.95, p.adj = "holm",
      digits = 2, p.digits = 3, as.na = NULL, plot = FALSE,
      geom = c("all", "point", "line", "ci", "jitter"),
      filename = NULL, width = NA, height = NA, dpi = 600,
      write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

formula	a formula of the form `cbind(time1, time2, time3) ~ 1` where `time1`, `time2`, and `time3` are numeric variables representing the levels of the within- subject factor, i.e., data are specified in wide-format (i.e., multivariate person level format).
data	a matrix or data frame containing the variables in the formula `formula`.
print	a character vector indicating which sphericity correction to use, i.e., `all` for all corrections, `none` for no correction, `GG` for Greenhouse-Geisser correction, `HF` for Huynh-Feldt correction, and `LB` for lower bound correction.
hypo	logical: if `TRUE` (default), null and alternative hypothesis are shown on the console.
epsilon	logical: if `TRUE` (default), box indices of sphericity (epsilon) are shown on the console, i.e., lower bound, Greenhouse and Geiser (GG), Huynh and Feldt (HF) and average of GG and HF.
descript	logical: if `TRUE` (default), descriptive statistics are shown on the console.
effsize	logical: if `TRUE`, effect size measures eta-squared (η^2), partial eta-squared (η_p^2), omega-squared (ω^2), and partial omega-squared (ω_p^2) for the repeated measures ANOVA and Cohen's *d* for the post hoc tests are shown on the console.
posthoc	logical: if `TRUE`, paired-samples t-tests for multiple comparison are conducted.
na.omit	logical: if `TRUE` (default), incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
adjust	logical: if `TRUE` (default), difference-adjustment for the Cousineau-Morey within-subject confidence intervals is applied.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
p.adj	a character string indicating an adjustment method for multiple testing based on `p.adjust`, i.e., `none`, `bonferroni`, `holm` (default), `h ochberg`, `hommel`, `BH`, `BY`, or `fdr`. See help page of the `p.adjust` function.
digits	an integer value indicating the number of decimal places to be used for displaying descriptive statistics and confidence interval.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
plot	logical: if `TRUE`, a plot showing error bars for confidence intervals is drawn. For additional plotting arguments, see `Details` in the help page of the function `plot.misty.object`
geom	a character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"point"` for drawing points representing means for each group, `"line"` for drawing a line connecting means of each group and lines connecting data points when requesting `"jitter"`, `"ci"` for drawing error bars representing confidence intervals, and `"jitter"` for drawing jittered data points with subject-specific lines when requesting `"ci"`.

filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

Sphericity The F-Test of the repeated measures ANOVA is based on the assumption of sphericity, which is defined as the assumption that the variance of differences between repeated measures are equal in the population. The Mauchly's test is commonly used to test this hypothesis. However, test of assumptions addresses an irrelevant hypothesis because what matters is the degree of violation rather than its presence (Baguley, 2012a). Moreover, the test is not recommended because it lacks statistical power (Abdi, 2010). Instead, the Box index of sphericity (ε) should be used to assess the degree of violation of the sphericity assumption. The ε parameter indicates the degree to which the population departs from sphericity with $\varepsilon = 1$ indicating that sphericity holds. As the departure becomes more extreme, ε approaches its lower bound $\hat{\varepsilon}_{lb}$:

$$\hat{\varepsilon}_{lb} = \frac{1}{J-1}$$

where J is the number of levels of the within-subject factor. Box (1954a, 1954b) suggested a measure for sphericity, which applies to a population covariance matrix. Greenhouse and Geisser (1959) proposed an estimate for ε known as $\hat{\varepsilon}_{gg}$ that can be computed from the sample covariance matrix, whereas Huynh and Feldt (1976) proposed an alternative estimate $\hat{\varepsilon}_{hf}$. These estimates can be used to correct the effect and error *df* of the *F*-test. Simulation studies showed that $\hat{\varepsilon}_{gg} \leq \hat{\varepsilon}_{hf}$ and that $\hat{\varepsilon}_{gg}$ tends to be conservative underestimating ε, whereas $\hat{\varepsilon}_{hf}$ tends to be liberal overestimating ε and occasionally exceeding one. Baguley (2012a) recommended to compute the average of the conservative estimate $\hat{\varepsilon}_{gg}$ and the liberal estimate $\hat{\varepsilon}_{hf}$ to assess the sphericity assumption. By default, the function prints results depending on the average $\hat{\varepsilon}_{gg}$ and $\hat{\varepsilon}_{hf}$:

- If the average is less than 0.75 results of the F-Test based on Greenhouse-Geiser correction factor ($\hat{\varepsilon}_{gg}$) is printed.
- If the average is less greater or equal 0.75, but less than 0.95 results of the F-Test based on Huynh-Feldt correction factor ($\hat{\varepsilon}_{hf}$) is printed.
- If the average is greater or equal 0.95 results of the F-Test without any corrections are printed.

Missing Data The function uses listwise deletion by default to deal with missing data. However, the function also allows to use all available observations by conducting the repeated measures ANOVA in long data format when specifying `na.omit = FALSE`. Note that in the presence of missing data, the F-Test without any sphericity corrections may be reliable, but it is not clear whether results based on Greenhouse-Geiser or Huynh-Feldt correction are trustworthy given that pairwise deletion is used for estimating the variance-covariance matrix when computing $\hat{\varepsilon}_{gg}$ and the total number of subjects regardless of missing values (i.e., complete and incomplete cases) are used for computing $\hat{\varepsilon}_{hf}$.

Within-Subject Confidence Intervals The function provides a plot showing error bars for difference-adjusted Cousineau-Morey confidence intervals (Baguley, 2012b). The intervals matches that of a CI for a difference, i.e., non-overlapping CIs corresponds to an inferences of no statistically significant difference. The Cousineau-Morey confidence intervals without adjustment can be used by specifying `adjust = FALSE`.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the data (`data`) in wide-format (`wide`), reshaped data in long-format (`long`), and within-subject confidence intervals (`ci`)
`formula`	formula of the current analysis
`args`	specification of function arguments
`plot`	ggplot2 object for plotting the results
`result`	list with result tables

Author(s)

Takuya Yanagida

References

Abdi, H. (2010). The Greenhouse-Geisser correction. In N. J. Salkind (Ed.) *Encyclopedia of Research Design* (pp. 630-634), Sage. https://dx.doi.org/10.4135/9781412961288

Baguley, T. S. (2012a). *Serious stats: A guide to advanced statistics for the behavioral sciences*. Palgrave Macmillan.

Baguley, T. (2012b). Calculating and graphing within-subject confidence intervals for ANOVA. *Behavior Research Methods, 44*, 158-175. https://doi.org/10.3758/s13428-011-0123-7

Bakerman, R. (2005). Recommended effect size statistics for repeated measures designs. *Behavior Research Methods*, 37, 179-384. https://doi.org/10.3758/BF03192707

Box, G. E. P. (1954a) Some Theorems on Quadratic Forms Applied in the Study of Analysis of Variance Problems, I. Effects of Inequality of Variance in the One-way Classification. *Annals of Mathematical Statistics, 25*, 290–302.

Box, G. E. P. (1954b) Some Theorems on Quadratic Forms Applied in the Study of Analysis of Variance Problems, II. Effects of Inequality of Variance and of Correlation between Errors in the Two-way Classification. *Annals of Mathematical Statistics, 25*, 484–98.

Greenhouse, S. W., and Geisser, S. (1959). On methods in the analysis of profile data.*Psychometrika, 24*, 95–112. https://doi.org/10.1007/BF02289823

Huynh, H., and Feldt, L. S. (1976). Estimation of the box correction for degrees of freedom from sample data in randomized block and splitplot designs. *Journal of Educational Statistics, 1*, 69–82. https://doi.org/10.2307/1164736

Olejnik, S., & Algina, J. (2000). Measures of effect size for comparative studies: Applications, interpretations, and limitations. *Contemporary Educational Psychology, 25*, 241–286. https://doi.org/10.1006/ceps.2000.1040

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

aov.b, test.t, test.z, cohens.d, ci.mean.diff, ci.mean

Examples

```
#————————————————————————————————————————————————————————————————————————
# Within-Subject Analysis of Variance

# Data frame
dat <- data.frame(time1 = c(3, 2, 1, 4, 5, 2, 3, 5, 6, 7),
                  time2 = c(4, 3, 6, 5, 8, 6, 7, 3, 4, 5),
                  time3 = c(1, 2, 2, 3, 6, 5, 1, 2, 4, 6))

# Example 1a: Within-Subject ANOVA, default setting
aov.w(cbind(time1, time2, time3) ~ 1, data = dat)

# Example 1b: Print descriptive statistics and Paired-samples t-tests
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, descript = TRUE, posthoc = TRUE)

# Example 1c: Print eta-squared and omega-squared
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, effsize = TRUE)

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 2a: Plot results, default setting
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, plot = TRUE)

# Example 2b: No bars, jittered data points without subject-specific lines
```

```
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, plot = TRUE, geom = "jitter")

# Example 2c: Plot results, additional arguments
# see Details in the help page of the function plot.misty.object
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, geom = "all", plot = TRUE,
      jitter.alpha = 0.4, title = "Within-Subject ANOVA")

# Example 2d: Plot results using the plot() function, additional arguments
# see Details in the help page of the function plot.misty.object
object <- aov.w(cbind(time1, time2, time3) ~ 1, data = dat)
plot(object, geom = c("point", "line", "jitter"), jitter.alpha = 0.4,
     title = "Within-Subject ANOVA")

#————————————————————————————————————————————————————————————————————————
# Create Plot Manually

# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- aov.w(cbind(time1, time2, time3) ~ 1, data = dat)

# Compute Means and difference-adjusted confidence intervals
ci.table <- ci.mean.w(object$data$wide, adjust = TRUE, output = FALSE)$result

# Example 3: Plot
ggplot(object$data$long, aes(time, y, group = 1)) +
  geom_errorbar(data = ci.table, aes(variable, m, ymin = low, ymax = upp),
                width = 0.1) +
  geom_point(data = ci.table, aes(variable, m), stat = "identity", size = 3) +
  geom_line(data = ci.table, aes(variable, m), stat = "identity") +
  geom_jitter(alpha = 0.2, width = 0.05, height = 0, size = 1.25) +
  theme_bw()

## Not run:
#————————————————————————————————————————————————————————————————————————
# Write Results and Save Plot

# Example 4a: Write results into a text file
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, write = "RM-ANOVA.txt")

# Example 4b: Write results into an Excel file
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, write = "RM-ANOVA.xlsx")

# Example 4c: Save plot as PNG fine
aov.w(cbind(time1, time2, time3) ~ 1, data = dat, plot = TRUE,
            filename = "RM-ANOVA.png", width = 6, height = 5)

## End(Not run)
```

---

blimp *Create, Run, and Print Blimp Models*

Description

This wrapper function creates a Blimp input file, runs the input file by using the `blimp.run()` function, and prints the Blimp output file by using the `blimp.print()` function.

Usage

```
blimp(x, file = "Blimp_Input.imp", data = NULL, comment = FALSE, replace.inp = TRUE,
      blimp.run = TRUE, posterior = FALSE, folder = "Posterior_",
      format = c("csv", "csv2", "excel", "rds", "workspace"), clear = TRUE,
      replace.out = c("always", "never", "modified"), Blimp = .detect.blimp(),
      result = c("all", "default", "algo.options", "data.info", "model.info",
                 "warn.mess", "error.mess", "out.model", "gen.param"),
      exclude = NULL, color = c("none", "blue", "green"),
      style = c("bold", "regular"), not.result = TRUE, write = NULL,
      append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`x`	a character string containing the Blimp input text.
`file`	a character string indicating the name of the Blimp input file with or without the file extension `.imp`, e.g., `"Blimp_Input.imp"` or `"Blimp_Input.imp"`.
`data`	a matrix or data frame from which the variables names for the section `VARIABLES` are extracted.
`comment`	logical: if `FALSE` (default), comments (i.e., text after the # symbol) are removed from the input text specified in the argument `x`.
`replace.inp`	logical: if `TRUE` (default), an existing input file will be replaced.
`blimp.run`	logical: if `TRUE`, the input file specified in the argument `file` containing the input text specified in the argument `x` is run using the `blimp.run()` function.
`posterior`	logical: if `TRUE`, the posterior distribution including burn-in and post-burn-in phase for all parameters are saved in long format in a file called `posterior.*` in the folder specified in the argument `folder` and `.imp` file name in the format specified in the argument `format`.
`folder`	a character string indicating the prefix of the folder for saving the posterior distributions. The default setting is `folder = "Posterior_"`.
`format`	a character vector indicating the file format(s) for saving the posterior distributions, i.e., `"csv"` (default) for `write.csv()`, `"csv2"` for `write.csv2()`, `"excel"` for `write.xlsx()`, `"rds"` for `saveRDS()`, and `"workspace"` for `write()`.
`clear`	logical: if `TRUE` (default), the console is cleared after estimating each model.
`replace.out`	a character string for specifying three settings: `"always"` (default), which runs all models, regardless of whether an output file for the model exists, `"never"`, which does not run any model that has an existing output file, and `"modified"`, which only runs a model if the modified date for the input file is more recent than the output file modified date.

Blimp	a character string for specifying the name or path of the Blimp executable to be used for running models. This covers situations where Blimp is not in the system's path, or where one wants to test different versions of the Blimp program. Note that there is no need to specify this argument for most users since it has intelligent defaults.
result	a character vector specifying Blimp result sections included in the output (see 'Details' in the blimp.print function).
exclude	a character vector specifying Blimp input command or result sections excluded from the output (see 'Details' in the blimp.print function).
color	a character vector with two elements indicating the colors used for the main headers (e.g., "ALGORITHMIC OPTIONS SPECIFIED:"), and for the headers Outcome Variable: and Missing predictor:, Latent Variable:, and Covariance Matrix:.
style	a character vector with two elements indicating the style used for headers (e.g., "ALGORITHMIC OPTIONS SPECIFIED:"), and for the main headers (e.g., "ALGORITHMIC OPTIONS SPECIFIED:"), and for the headers Outcome Variable: and Missing predictor:, Complete variable:, Latent Variable:, and Covariance Matrix:.
not.result	logical: if TRUE (default), character vector indicating the result sections not requested are shown on the console.
write	a character string naming a file for writing the output into a text file with file extension ".txt" (e.g., "Output.txt").
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console by using the function blimp.print().

Details

VARIABLES **Section** The VARIABLES section used to assign names to the variables in the data set can be specified by using the data argument:

- Write Blimp Data File: In the first step, the Blimp data file is written by using the write.mplus() function, e.g. write.mplus(data1, file = "data1.dat").
- Specify Blimp Input: In the second step, the Blimp input is specified as a character string. The VARIABLES option is left out from the Blimp input text, e.g., input <- 'DATA: data1.dat;\nMODEL: y ~ x1@b1 x2@b2 d2;'.
- Run Blimp Input: In the third step, the Blimp input is run by using the blimp() function. The argument data needs to be specified given that the VARIABLES section was left out from the Blimp input text in the previous step, e.g., blimp(input, file = "Ex4.3.imp", data = data1).

Note that unlike Mplus, Blimp allows to specify a CSV data file with variable names in the first row. Hence, it is recommended to export the data from R using the write.csv() function to specify the data file in the DATA section of the Blimp input file without specifying the VARIABLES section.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	a character vector containing the Blimp input text
`args`	specification of function arguments
`write`	write command sections
`result`	list with result sections (`result`)

Author(s)

Takuya Yanagida

References

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3). Retrieved from www.appliedmissingdata.com/blimp

See Also

`blimp.update`, `blimp.run`, `blimp.print`, `blimp.plot`, `blimp.bayes`

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Example 1: Write data, specify input without VARIABLES section, and run input

# Write Data File
# Note that row.names = FALSE needs to be specified
write.csv(data1, file = "data1.csv", row.names = FALSE)

# Specify Blimp input
input1 <- '
DATA: data1.csv;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
'

# Run Blimp input
blimp(input1, file = "Ex4.3.imp")

#————————————————————————————————————————————————————————————————————————
```

```
# Example 2: Write data, specify input with VARIABLES section, and run input

# Write Data File
write.mplus(data1, file = "data1.dat", input = FALSE)

# Specify Blimp input
input2 <- '
DATA: data1.dat;
VARIABLES: id v1 v2 v3 y x1 d x2 v4;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
'

# Run Blimp input
blimp(input2, file = "Ex4.3.imp")

#————————————————————————————————————————————————————————————————————————
# Example 3: Alternative specification using the data argument

# Write Data File
write.mplus(data1, file = "data1.dat", input = FALSE)

# Specify Blimp input
input3 <- '
DATA: data1.dat;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
'

# Run Blimp input
blimp(input3, file = "Ex4.3.imp", data = data1)

## End(Not run)
```

`blimp.bayes` *Blimp Summary Measures, Convergence and Efficiency Diagnostics*

Description

This function reads the posterior distribution for all parameters saved in long format in a file called `posterior.*` by the function `blimp.run` or `blimp` when specifying `posterior = TRUE` to compute point estimates (i.e., mean, median, and MAP), measures of dispersion (i.e., standard deviation and mean absolute deviation), measures of shape (i.e., skewness and kurtosis), credible intervals (i.e., equal-tailed intervals and highest density interval), convergence and efficiency diagnostics (i.e., potential scale reduction factor R-hat, effective sample size, and Monte Carlo standard error), probability of direction, and probability of being in the region of practical equivalence for the posterior distribution for each parameter. By default, the function computes the maximum of rank-normalized split-R-hat and rank normalized folded-split-R-hat, Bulk effective sample size (Bulk-ESS) for rank-normalized values using split chains, tail effective sample size (Tail-ESS) defined as the minimum of the effective sample size for 0.025 and 0.975 quantiles, the Bulk Monte Carlo standard error (Bulk-MCSE) for the median and Tail Monte Carlo standard error (Tail-MCSE) defined as the maximum of the MCSE for 0.025 and 0.975 quantiles.

Usage

```
blimp.bayes(x, param = NULL,
            print = c("all", "default", "m", "med", "map", "sd", "mad",
                      "skew", "kurt", "eti", "hdi",
                      "rhat", "b.ess", "t.ess", "b.mcse", "t.mcse"),
            m.bulk = FALSE, split = TRUE, rank = TRUE, fold = TRUE,
            pd = FALSE, null = 0, rope = NULL,
            ess.tail = c(0.025, 0.975), mcse.tail = c(0.025, 0.975),
            alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
            digits = 2, r.digits = 3, ess.digits = 0, mcse.digits = 3,
            p.digits = 3, write = NULL, append = TRUE, check = TRUE,
            output = TRUE)
```

Arguments

`x` a character string indicating the name of folder containing the `posterior.*` file, e.g., `"Posterior_Ex4.3"` or the name of the `posterior.*` file with or without any file extension, e.g., `"Posterior_ExEx4.3/posterior.csv"` or `"Posterior_ExEx4.3/posterior"`. Alternatively, a `misty.object` of type `blimp` can be specified, i.e., result object of the `blimp.plot()` function. Note that if the `posterior` file is specified without file extension while multiple `posterior.*` files in different file formats are available, then the file is read in following order: `csv`,`RData`, `rds`, and `xlsx`.

`param` a numeric vector indicating which parameters to print. Note that the number of the parameter (`Param`) and the parameter specification (`L1`, `L2`, and `L3`) are provided in the text file `"partable.txt"`.

`print` a character vector indicating which summary measures, convergence, and efficiency diagnostics to be printed on the console, i.e. `"all"` for all summary measures, convergence, and efficiency diagnostics, `"m"` for the mean, `"med"` for the median, `"MAP"` for the maximum a posteriori probability estimate, `"med"` for the standard deviation, `"mad"` for the mean absolute deviation, `"skew"` for the skewness, `"kurt"` for the kurtosis, `"eti"` for the equal-tailed credible interval,

"hdi" for the highest density credible interval, "rhat" for the potential scale reduction (PSR) factor R-hat convergence diagnostic, "b.ess" for the bulk effective sample size (ESS), "t.ess" for the tail ESS, "b.mcse" for the bulk Monte Carlo standard error (MCSE), and "t.mcse" for the tail MCSE. The default setting is print = c("med", "sd", "skew", "kurt", "eti", "rhat", "b.ess", "t.ess", "b.mcse", "t.mcse").

m.bulk — logical: if TRUE the Monte Carlo standard error for the mean is computed. The default setting is m.bulk = FALSE, i.e., the Monte Carlo standard error for the median is computed.

split — logical: if TRUE (default), each MCMC chain is split in half before computing R-hat. Note that the argument split is always set to FALSE when computing ESS.

rank — logical: if TRUE (default), rank-normalization is applied to the posterior draws before computing R-hat and ESS. Note that the argument rank is always set to FALSE when computing MCSE.

fold — logical: if TRUE (default), the maximum of rank-normalized split-R-hat and rank normalized folded-split-R-hat is computed. Note that the arguments split and rank are always set to TRUE when specifying fold = TRUE.

pd — logical: if TRUE, the probability of direction is printed on the console.

null — a numeric value considered as a null effect for the probability of direction (default is 0). Note that the value specified in the argument null applies to all parameters which might not be sensible for all parameters.

rope — a numeric vector with two elements indicating the ROPE's lower and upper bounds. ROPE is also depending on the argument alternative, e.g., if rope = c(-0.1, 0.1), then the actual ROPE is [-0.1, 0.1] given alternative = "two.sided} (default), \code{[-Inf, 0.1]} given \code{alternative = "greater, and [-0.1, Inf] given alternative = "less". Note that the interval specified in the argument rope applies to all parameters which might not be sensible for all parameters.

ess.tail — a numeric vector with two elements to specify the quantiles for computing the tail ESS. The default setting is tail = c(0.025, 0.975), i.e., tail ESS is the minimum of effective sample sizes for 0.025 and 0.975 quantiles.

mcse.tail — a numeric vector with two elements to specify the quantiles for computing the tail MCSE. The default setting is tail = c(0.025, 0.975), i.e., tail MCSE is the maximum of Monte Carlo standard error for 0.025 and 0.975 quantiles.

alternative — a character string specifying the alternative hypothesis for the credible intervals, must be one of "two.sided" (default), "greater" or "less".

conf.level — a numeric value between 0 and 1 indicating the confidence level of the credible interval. The default setting is conf.level = 0.95.

digits — an integer value indicating the number of decimal places to be used for displaying point estimates, measures of dispersion, and credible intervals.

r.digits — an integer value indicating the number of decimal places to be used for displaying R-hat values.

ess.digits — an integer value indicating the number of decimal places to be used for displaying effective sample sizes.

`mcse.digits` an integer value indicating the number of decimal places to be used for displaying Monte Carlo standard errors.

`p.digits` an integer value indicating the number of decimal places to be used for displaying the probability of direction and the probability of being in the region of practical equivalence (ROPE).

`write` a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.

`append` logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.

`check` logical: if `TRUE` (default), argument specification is checked.

`output` logical: if `TRUE` (default), output is shown on the console by using the function `blimp.print()`.

Details

Convergence and Efficiency Diagnostics for Markov Chains Convergence and efficiency diagnostics for Markov chains is based on following numeric measures:

- **Potential Scale Reduction (PSR) factor R-hat**: The PSR factor R-hat compares the between- and within-chain variance for a model parameter, i.e., R-hat larger than 1 indicates that the between-chain variance is greater than the within-chain variance and chains have not mixed well. According to the default setting, the function computes the improved R-hat as recommended by Vehtari et al. (2020) based on rank-normalizing (i.e., `rank = TRUE`) and folding (i.e., `fold = TRUE`) the posterior draws after splitting each MCMC chain in half (i.e., `split = TRUE`). The traditional R-hat used in Blimp can be requested by specifying `split = TRUE`, `rank = FALSE`, and `fold = FALSE`. Note that the traditional R-hat can catch many problems of poor convergence, but fails if the chains have different variances with the same mean parameter or if the chains have infinite variance with one of the chains having a different location parameter to the others (Vehtari et al., 2020). According to Gelman et al. (2014) a R-hat value of 1.1 or smaller for all parameters can be considered evidence for convergence. The Stan Development Team (2024) recommends running at least four chains and a convergence criterion of less than 1.05 for the maximum of rank normalized split-R-hat and rank normalized folded-split-R-hat. Vehtari et al. (2020), however, recommended to only use the posterior samples if R-hat is less than 1.01 because the R-hat can fall below 1.1 well before convergence in some scenarios (Brooks & Gelman, 1998; Vats & Knudon, 2018).
- **Effective Sample Size (ESS)**: The ESS is the estimated number of independent samples from the posterior distribution that would lead to the same precision as the autocorrelated samples at hand. According to the default setting, the function computes the ESS based on rank-normalized split-R-hat and within-chain autocorrelation. The function provides the estimated Bulk-ESS (`B.ESS`) and the Tail-ESS (`T.ESS`). The Bulk-ESS is a useful measure for sampling efficiency in the bulk of the distribution (i.e, efficiency of the posterior mean), and the Tail-ESS is useful measure for sampling efficiency in the tails of the distribution (e.g., efficiency of tail quantile estimates). Note that by default, the Tail-ESS is the minimum of the effective sample sizes for 2.5% and 97.5% quantiles (`tail = c(0.025, 0.975)`). According to Kruschke (2015), a rank-normalized ESS greater

than 400 is usually sufficient to get a stable estimate of the Monte Carlo standard error. However, a ESS of at least 1000 is considered optimal (Zitzmann & Hecht, 2019).

- **Monte Carlo Standard Error (MCSE)**: The MCSE is defined as the standard deviation of the chains divided by their effective sample size and reflects uncertainty due to the stochastic algorithm of the Markov Chain Monte Carlo method. The function provides the estimated Bulk-MCSE (`B.MCSE`) for the margin of error when using the MCMC samples to estimate the posterior mean and the Tail-ESS (`T.MCSE`) for the margin of error when using the MCMC samples for interval estimation.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	a character string indicating the name of the `posterior.*`
`args`	specification of function arguments
`data`	posterior distribution of each parameter estimate in long format
`result`	result table with summary measures, convergence, and efficiency diagnostics

Note

This function is a modified copy of functions provided in the **rstan** package by Stan Development Team (2024) and **bayestestR** package by Makowski et al. (2019).

Author(s)

Takuya Yanagida

References

Brooks, S. P. and Gelman, A. (1998). General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics, 7*(4): 434–455. MR1665662.

Gelman, A., & Rubin, D.B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science, 7*, 457-472. https://doi.org/10.1214/ss/1177011136

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3).
Retrieved from www.appliedmissingdata.com/blimp

Kruschke, J. (2015). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

Makowski, D., Ben-Shachar, M., & Lüdecke, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the Bayesian framework. *Journal of Open Source Software, 4*(40), 1541. https://doi.org/10.21105/joss.01541

Stan Development Team (2024). *RStan: the R interface to Stan*. R package version 2.32.6. https://mc-stan.org/.

Vats, D. and Knudson, C. (2018). Revisiting the Gelman-Rubin Diagnostic. arXiv:1812.09384.

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2020). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian analysis, 16*(2), 667-718. https://doi.org/110.1214/20-BA1221

Zitzmann, S., & Hecht, M. (2019). Going beyond convergence in Bayesian estimation: Why precision matters too and how to assess it. *Structural Equation Modeling, 26*(4), 646–661. https://doi.org/10.1080/10705511.2018.1545232

See Also

blimp, blimp.update, blimp.run, blimp.plot,blimp.print, blimp.plot,

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Blimp Example 4.3: Linear Regression

# Example 1a: Default setting, specifying name of the folder
blimp.bayes("Posterior_Ex4.3")

# Example 1b: Default setting, specifying the posterior file
blimp.bayes("Posterior_Ex4.3/posterior.csv")

# Example 2a: Print all summary measures, convergence, and efficiency diagnostics
blimp.bayes("Posterior_Ex4.3", print = "all")

# Example 3a: Print default measures plus MAP
blimp.bayes("Posterior_Ex4.3", print = c("default", "map"))

# Example 4: Print traditional R-hat in line with Blimp
blimp.bayes("Posterior_Ex4.3", split = TRUE, rank = FALSE, fold = FALSE)

# Example 5: Print probability of direction and the probability of
# being ROPE [-0.1, 0.1]
blimp.bayes("Posterior_Ex4.3", pd = TRUE, rope = c(-0.1, 0.1))

# Example 6: Write Results into a text file
blimp.bayes("Posterior_Ex4.3", write = "Bayes_Summary.txt")

# Example 7b: Write Results into an Excel file
blimp.bayes("Posterior_Ex4.3", write = "Bayes_Summary.xlsx")

## End(Not run)
```

---

blimp.plot *Blimp Trace Plots and Posterior Distribution Plots*

Description

This function reads the posterior distribution including burn-in and post-burn-in phase for all parameters saved in long format in a file called `posterior.*` by the function `blimp.run` or `blimp` when specifying `posterior = TRUE` to display trace plots and posterior distribution plots.

Usage

```
blimp.plot(x, plot = c("none", "trace", "post"), param = NULL, labels = TRUE,
           burnin = TRUE, point = c("all", "none", "m", "med", "map"),
           ci = c("none", "eti", "hdi"), conf.level = 0.95, hist = TRUE,
           density = TRUE, area = TRUE, alpha = 0.4, fill = "gray85",
           facet.nrow = NULL, facet.ncol = NULL,
           facet.scales = c("fixed", "free", "free_x", "free_y"),
           xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
           xbreaks = ggplot2::waiver(), ybreaks = ggplot2::waiver(),
           xexpand = ggplot2::waiver(), yexpand = ggplot2::waiver(),
           palette = "Set 2", binwidth = NULL, bins = NULL,
           density.col = "#0072B2", shape = 21,
           point.col = c("#CC79A7", "#D55E00", "#009E73"),
           linewidth = 0.6, linetype = "dashed", line.col = "black",
           plot.margin = NULL, legend.title.size = 10, legend.text.size = 10,
          legend.box.margin = NULL, saveplot = c("all", "none", "trace", "post"),
           filename = "Blimp_Plot.pdf", file.plot = c("_TRACE", "_POST"),
          width = NA, height = NA, units = c("in", "cm", "mm", "px"), dpi = 600,
           check = TRUE)
```

Arguments

`x`	a character string indicating the name of folder containing the `posterior.*` file, e.g., `"Posterior_Ex4.3"` or the name of the `posterior.*` file with or without any file extension, e.g., `"Posterior_ExEx4.3/posterior.csv"` or `"Posterior_ExEx4.3/posterior"`. Alternatively, a `misty.object` of type `blimp` can be specified, i.e., result object of the `blimp.plot()` function. Note that if the `posterior` file is specified without file extension while multiple `posterior.*` files in different file formats are available, then the file is read in following order: `csv`,`RData`, `rds`, and `xlsx`.
`plot`	a character string indicating the type of plot to display, i.e., `"none"` for not displaying any plot, `"trace"` (default) for displaying trace plots, and `post` for displaying posterior distribution plots.
`param`	a numeric vector indicating which parameters to print for the trace plots or posterior distribution plots. Note that the number of the parameter (`Param`) and the parameter specification (`L1`, `L2`, and `L3`) are provided in the text file `"partable.txt"`. Note that parameters with zero variance are excluded by default.
`labels`	logical: if `TRUE` (default), parameter labels (e.g., `y Beta x` for the slope of the regression y on x) are shown in the facet labels. If `FALSE`, the numbers of the parameter (e.g., `Parameter 1` are shown in the the facet labels.

burnin — logical: if FALSE, the burn-in iterations are discarded when displaying trace plots. The default setting for plot = "trace" is TRUE. Note that the burn-in iterations are always discarded when displaying posterior distribution plots (plot = "post") regardless of the setting of the argument burnin.

point — a character vector indicating the point estimate(s) to be displayed in the posterior distribution plots, i.e., "all" for all point estimates, "none" for not displaying any point estimates, "m" for the posterior mean estimate, "med" (default) for the posterior median estimate, and "map" for the maximum a posterior estimate.

ci — a character string indicating the type of credible interval to be displayed in the posterior distribution plots, i.e., "none" for not displaying any credible intervals, "eti" (default) for displaying the equal-tailed intervals and "hdi" for displaying the highest density interval.

conf.level — a numeric value between 0 and 1 indicating the confidence level of the credible interval (default is 0.95).

hist — logical: if TRUE (default), histograms are drawn in the posterior probability plots.

density — logical: if TRUE (default), density curves are drawn in the posterior probability plots.

area — logical: if TRUE (default), statistical not significant and statistical significant area is filled with a different color and vertical lines are drawn.

alpha — a numeric value between 0 and 1 for the alpha argument (default is 0.4) for the annotate, and geom_histogram function.

fill — a character string indicating the color for the "fill" argument (default is "gray85") for the annotate and geom_histogram functions.

facet.nrow — a numeric value indicating the nrow argument (default is NULL) for the facet_wrap function.

facet.ncol — a numeric value indicating the ncol argument (default is 2) for the facet_wrap function.

facet.scales — a character string indicating the scales argument (default is "free") for the facet_wrap function.

xlab — a character string indicating the name argument for the scale_x_continuous function.

ylab — a character string indicating the name argument for the scale_y_continuous function.

xlim — a numeric vector with two elements indicating the limits argument (default it NULL) for the scale_x_continuous function.

ylim — a numeric vector with two elements indicating the limits argument (default it NULL) for the scale_y_continuous function.

xbreaks — a numeric vector indicating the breaks argument (default is ggplot2::waiver()) for the scale_x_continuous function.

ybreaks — a numeric vector indicating the breaks argument (default is ggplot2::waiver()) for the scale_y_continuous function.

xexpand — a numeric vector with two elements indicating the expand argument (default is (0.02, 0)) for the scale_x_continuous function.

yexpand
: a numeric vector with two elements indicating the expand argument for the scale_y_continuous function. Note that the default setting depends on the type of plot, e.g., (0.02, 0) for the trace plots and expansion(mult = c(0, 0.05)) for the posterior distribution plots.

palette
: a character string indicating the palette name (default is "Set 2") for the hcl.colors function. Note that the character string must be one of hcl.pals().

binwidth
: a numeric value indicating the binwidth argument (default is to use the number of bins in bins argument) for the geom_histogram function.

bins
: a numeric value indicating the bins argument (default is 30) for the geom_histogram function.

density.col
: a character string indicating the color argument (default is "#0072B2") for the geom_density function.

shape
: a numeric value indicating the shape argument (default is 21) for the geom_point function.

point.col
: a character vector with three elements indicating the values argument (default is c("#CC79A7", "#D55E00", "#009E73")) for the scale_color_manual function.

linewidth
: a numeric value indicating the linewidth argument (default is 0.6) for the geom_vline function.

linetype
: a numeric value indicating the linetype argument (default is "dashed") for the geom_vline function.

line.col
: a character string indicating the color argument (default is "black") for the geom_vline function.

plot.margin
: a numeric vector indicating the plot.margin argument for the theme function. Note that the default setting depends on the type of the plot, e.g., c(4, 15, -10, 0) for the trace plots, and c(4, 15, 4, 4) for the autocorrelation plots.

legend.title.size
: a numeric value indicating the legend.title argument (default is element_text(size = 10)) for the theme function.

legend.text.size
: a numeric value indicating the legend.text argument (default is element_text(size = 10)) for the theme function.

legend.box.margin
: a numeric vector indicating the legend.box.margin argument for the theme function. Note that the default setting depends on the type of plot, e.g., c(-16, 6, 6, 6) for the trace plots, and c(-25, 6, 6, 6) for the posterior distribution plots with displaying point estimates.

saveplot
: a character vector indicating the plot to be saved, i.e., "all" for saving all plots, "none" (default) for not saving any plots, "trace" for saving the trace plots and post for the saving the posterior distribution plots.

filename
: a character string indicating the filename argument (default is "Blimp_Plot.pdf") including the file extension for the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument.

file.plot	a character vector with two elements for distinguishing different types of plots. By default, the character string specified in the argument "filename" ("Blimp_Plot") is concatenated with "_TRACE" ("Blimp_Plot_TRACE") for the trace plots, and "_POST" ("Blimp_Plot_POST") for the posterior distribution plots.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
units	a character string indicating the units argument (default is in) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	a character string indicating the name of the posterior.* file
args	specification of function arguments
data	list with posterior distribution of each parameter estimate in long format (plotdat), plot data for the trace plots (trace), and plot data for the posterior distribution plots (post).
plot	list with the trace plots (trace and posterior distribution plots (post)

Author(s)

Takuya Yanagida

References

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3). Retrieved from www.appliedmissingdata.com/blimp

See Also

blimp, blimp.update, blimp.run, blimp.print, blimp.plot, blimp.bayes

Examples

```
## Not run:

#-----------------------------------------------------------------------
# Blimp Example 4.3: Linear Regression

#...................
```

```
# Trace Plots

# Example 1a: Default setting, specifying name of the folder
blimp.plot("Posterior_Ex4.3")

# Example 1b: Default setting, specifying the posterior file
blimp.plot("Posterior_Ex4.3/posterior.csv")

# Example 1c: Print parameters 2, 3, 4, and 5
blimp.plot("Posterior_Ex4.3", param = 2:5)

# Example 1e: Arrange panels in three columns
blimp.plot("Posterior_Ex4.3", ncol = 3)

# Example 1f: Specify "Pastel 1" palette for the hcl.colors function
blimp.plot("Posterior_Ex4.3", palette = "Pastel 1")

#···................
# Posterior Distribution Plots

# Example 2a: Default setting, i.e., posterior median and equal-tailed interval
blimp.plot("Posterior_Ex4.3", plot = "post")

# Example 2b: Display posterior mean and maximum a posteriori
blimp.plot("Posterior_Ex4.3", plot = "post", point = c("m", "map"))

# Example 2c: Display maximum a posteriori and highest density interval
blimp.plot("Posterior_Ex4.3", plot = "post", point = "map", ci = "hdi")

# Example 2d: Do not display any point estimates and credible interval
blimp.plot("Posterior_Ex4.3", plot = "post", point = "none", ci = "none")

# Example 2d: Do not display histograms
blimp.plot("Posterior_Ex4.3", plot = "post", hist = FALSE)

#···................
# Save Plots

# Example 3a: Save all plots in pdf format
blimp.plot("Posterior_Ex4.3", saveplot = "all")

# Example 3b: Save all plots in png format with 300 dpi
blimp.plot("Posterior_Ex4.3", saveplot = "all", filename = "Blimp_Plot.png", dpi = 300)

# Example 3a: Save posterior distribution plot, specify width and height of the plot
blimp.plot("Posterior_Ex4.3", plot = "none", saveplot = "post",
           width = 7.5, height = 7)

#————————————————————————————————————————————————————————————————————————
# Plot from misty.object

# Create misty.object
object <- blimp.plot("Posterior_Ex4.3", plot = "none")
```

```
# Trace plot
blimp.plot(object, plot = "trace")

# Posterior distribution plot
blimp.plot(object, plot = "post")

#————————————————————————————————————————————————————————————————————————
# Create Plots Manually

# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- blimp.plot("Posterior_Ex4.3", plot = "none")

#···................
# Example 4: Trace Plots

# Extract data
data.trace <- object$data$trace

# Plot
ggplot(data.trace, aes(x = iter, y = value, color = chain)) +
  annotate("rect", xmin = 0, xmax = 1000, ymin = -Inf, ymax = Inf,
           alpha = 0.4, fill = "gray85") +
  geom_line() +
  facet_wrap(~ param, ncol = 2, scales = "free") +
  scale_x_continuous(name = "", expand = c(0.02, 0)) +
  scale_y_continuous(name = "", expand = c(0.02, 0)) +
  scale_colour_manual(name = "Chain",
                      values = hcl.colors(n = 2, palette = "Set 2")) +
  theme_bw() +
  guides(color = guide_legend(nrow = 1, byrow = TRUE)) +
  theme(plot.margin = margin(c(4, 15, -10, 0)),
        legend.position = "bottom",
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 10),
        legend.box.margin = margin(c(-16, 6, 6, 6)),
        legend.background = element_rect(fill = "transparent"))

#···................
# Example 5: Posterior Distribution Plots

# Extract data
data.post <- object$data$post

# Plot
ggplot(data.post, aes(x = value)) +
  geom_histogram(aes(y = after_stat(density)), color = "black", alpha = 0.4,
                 fill = "gray85") +
  geom_density(color = "#0072B2") +
  geom_vline(data = data.frame(param = levels(data.post$param),
```

```
                              stat = tapply(data.post$value, data.post$param, median)),
            aes(xintercept = stat, color = "Median"), linewidth = 0.6) +
  geom_vline(data = data.frame(param = levels(data.post$param),
                              low = tapply(data.post$value, data.post$param,
                                           function(y) quantile(y, probs = 0.025))),
            aes(xintercept = low), linetype = "dashed", linewidth = 0.6) +
  geom_vline(data = data.frame(param = levels(data.post$param),
                              upp = tapply(data.post$value, data.post$param,
                                           function(y) quantile(y, probs = 0.975))),
            aes(xintercept = upp), linetype = "dashed", linewidth = 0.6) +
  facet_wrap(~ param, ncol = 2, scales = "free") +
  scale_x_continuous(name = "", expand = c(0.02, 0)) +
  scale_y_continuous(name = "Probability Density, f(x)",
                   expand = expansion(mult = c(0L, 0.05))) +
  scale_color_manual(name = "Point Estimate", values = c(Median = "#D55E00")) +
  labs(caption = "95% Equal-Tailed Interval") +
  theme_bw() +
  theme(plot.margin = margin(4, 15, -8, 4),
        plot.caption = element_text(hjust = 0.5, vjust = 7),
        legend.position = "bottom",
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 10),
        legend.box.margin = margin(-30, 6, 6, 6),
        legend.background = element_rect(fill = "transparent"))

## End(Not run)
```

`blimp.print` *Print Blimp Output*

Description

This function prints the result sections of a Blimp output file (`.blimp-out`) on the R console. By default, the function prints selected result sections, i.e., `Algorithmic Options Specified`, `Data Information`, `Model Information`, `Warning Messages`, `Outcome Model Estimates`, and `Generated Parameters`.

Usage

```
blimp.print(x,
            result = c("all", "default", "algo.options", "data.info",
            "model.info", "warn.mess", "error.mess", "out.model", "gen.param"),
            exclude = NULL, color = c("none", "blue", "green"),
            style = c("bold", "regular"), not.result = TRUE,
            write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

x	a character string indicating the name of the Blimp output file with or without the file extension `.blimp-out`, e.g., `"Blimp_Output.blimp-out"` or `"Blimp_Output"`. Alternatively, a `misty.object` of type `blimp` can be specified, i.e., result object of the `blimp.print()` function.
result	a character vector specifying Blimp result sections included in the output (see 'Details').
exclude	a character vector specifying Blimp input command or result sections excluded from the output (see 'Details').
color	a character vector with two elements indicating the colors used for the main headers (e.g., `"ALGORITHMIC OPTIONS SPECIFIED:"`), and for the headers `Outcome Variable:` and `Missing predictor:`, `Latent Variable:`, and `Covariance Matrix:`.
style	a character vector with two elements indicating the style used for headers (e.g., `"ALGORITHMIC OPTIONS SPECIFIED:"`), and for the main headers (e.g., `"ALGORITHMIC OPTIONS SPECIFIED:"`), and for the headers `Outcome Variable:` and `Missing predictor:`, `Complete variable:`, `Latent Variable:`, and `Covariance Matrix:`.
not.result	logical: if `TRUE` (default), character vector indicating the result sections not requested are shown on the console.
write	a character string naming a file for writing the output into a text file with file extension `".txt"` (e.g., `"Output.txt"`).
append	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
check	logical: if `TRUE` (default), argument specification is checked.
output	logical: if `TRUE` (default), output is shown on the console.

Details

Result Sections Following result sections can be selected by using the `result` argument or excluded by using the `exclude` argument:

- `"algo.options"` for the `ALGORITHMIC OPTIONS SPECIFIED` section
- `"simdat.summary"` for the `SIMULATED DATA SUMMARIES` section
- `"order.simdat"` for the `VARIABLE ORDER IN SIMULATED DATA` section
- `"burnin.psr"` for the `BURN-IN POTENTIAL SCALE REDUCTION (PSR) OUTPUT` section
- `"mh.accept"` for the `METROPOLIS-HASTINGS ACCEPTANCE RATES` section
- `"data.info"` for the `DATA INFORMATION` section
- `"var.imp"` for the `VARIABLES IN IMPUTATION MODEL` section
- `"model.info"` for the `MODEL INFORMATION` section
- `"param.label"` for the `PARAMETER LABELS` section
- `"warn.mess"` for the `WARNING MESSAGES` section
- `"fit"` for the `MODEL FIT` section
- `"cor.resid"` for the `CORRELATIONS AMONG RESIDUALS` section
- `"out.model"` for the `OUTCOME MODEL ESTIMATES` section
- `"pred.model"` for the `PREDICTOR MODEL ESTIMATES` section

- "gen.param" for the GENERATED PARAMETERS section
- "order.impdat" for the VARIABLE ORDER IN IMPUTED DATA section

Note that all result sections are requested by specifying result = "all". The result argument is also used to select one (e.g., result = "algo.options") or more than one result sections (e.g., result = c("algo.options", "fit")), or to request result sections in addition to the default setting (e.g., result = c("default", "fit")). The exclude argument is used to exclude result sections from the output (e.g., exclude = "algo.options").

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	character string or misty object
args	specification of function arguments
print	print objects
notprint	character vectors indicating the result sections not requested
result	list with Blimp version (blimp) and result sections (result)

Author(s)

Takuya Yanagida

References

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3). Retrieved from www.appliedmissingdata.com/blimp

See Also

blimp, blimp.update, blimp.run, blimp.plot, blimp.bayes

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Blimp Example 4.3: Linear Regression

# Example 1a: Default setting
blimp.print("Ex4.3.blimp-out")

# Example 1c: Print OUTCOME MODEL ESTIMATES only
blimp.print("Ex4.3.blimp-out", result = "out.model")

# Example 1d: Print MODEL FIT in addition to the default setting
blimp.print("Ex4.3.blimp-out", result = c("default", "fit"))

# Example 1e: Exclude DATA INFORMATION section
```

```
blimp.print("Ex4.3.blimp-out", exclude = "data.info")

# Example 1f: Print all result sections, but exclude MODEL FIT section
blimp.print("Ex4.3.blimp-out", result = "all", exclude = "fit")

# Example 1g: Print result section in a different order
blimp.print("Ex4.3.blimp-out", result = c("model.info", "fit", "algo.options"))

#-----------------------------------------------------------------------
# misty.object of type 'blimp.print'

# Example 2
# Create misty.object
object <- blimp.print("Ex4.3.blimp-out", output = FALSE)

# Print misty.object
blimp.print(object)

#-----------------------------------------------------------------------
# Write Results

# Example 3: Write Results into a text file
blimp.print("Ex4.3.blimp-out", write = "Output_4-3.txt")

## End(Not run)
```

`blimp.run` *Run Blimp Models*

Description

This function runs a group of Blimp models (`.imp` files) located within a single directory or nested within subdirectories.

Usage

```
blimp.run(target = getwd(), recursive = FALSE,
          replace.out = c("always", "never", "modified"), posterior = FALSE,
        folder = "Posterior_", format = c("csv", "csv2", "xlsx", "rds", "RData"),
          clear = FALSE, Blimp = .detect.blimp(), check = TRUE)
```

Arguments

`target`	a character string indicating the directory containing Blimp input files (`.imp`) to run, a character string indicating a single `.imp` file to run, or a character vector for multiple `.imp` files to run. May be a full path, relative path, a file name, or a vector of file names within the working directory.
`recursive`	logical: if `TRUE`, run all models nested in subdirectories within a directory. Not relevant if a single or multiple `.imp` files were specified for the argument `target`.

`replace.out` a character string for specifying three settings: `"always"` (default), which runs all models, regardless of whether an output file for the model exists, `"never"`, which does not run any model that has an existing output file, and `"modified"`, which only runs a model if the modified date for the input file is more recent than the output file modified date.

`posterior` logical: if `TRUE`, the posterior distribution including burn-in and post-burn-in phase for all parameters are saved in long format in a file called `posterior.*` in the folder specified in the argument `folder` and `.imp` file name in the format specified in the argument `format`.

`folder` a character string indicating the prefix of the folder for saving the posterior distributions. The default setting is `folder = "Posterior_"`.

`format` a character vector indicating the file format(s) for saving the posterior distributions, i.e., `"csv"` (default) for `write.csv()`, `"csv2"` for `write.csv2()`, `"xlsx"` for `write.xlsx()`, `"rds"` for `saveRDS()`, and `"RData"` for `write()`.

`clear` logical: if `TRUE`, the console is cleared after estimating each model.

`Blimp` a character string for specifying the name or path of the Blimp executable to be used for running models. This covers situations where Blimp is not in the system's path, or where one wants to test different versions of the Blimp program. Note that there is no need to specify this argument for most users since it has intelligent defaults.

`check` logical: if `TRUE` (default), argument specification is checked.

Value

None.

Note

This function is based on the `detect_blimp()` and `rblimp()` function in the **rblimp** package by Brian T.Keller (2024).

Author(s)

Takuya Yanagida

References

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3). Retrieved from www.appliedmissingdata.com/blimp

Keller B (2024). *rblimp: Integration of Blimp Software into R*. R package version 0.1.31. https://github.com/blimp-stats/rblimp

See Also

`blimp`, `blimp.update`, `blimp.print`, `blimp.plot`, `blimp.bayes`

Examples

```
## Not run:

# Example 1: Run Blimp models located within the current working directory
blimp.run()

# Example 2: Run Blimp models located nested within subdirectories
blimp.run(recursive = TRUE)

# Example 3: Run Blimp input file
blimp.run("Ex4.1a.imp")

# Example 4: Run Blimp input files
blimp.run(c("Ex4.1a.imp", "Ex4.1b.imp"))

# Example 5: Run Blimp models, save posterior distribution in a R workspace
blimp.run(posterior = TRUE, format = "workspace")

## End(Not run)
```

`blimp.update` *Blimp Input Updating*

Description

This function updates specific input command sections of a `misty.object` of type `blimp` to create an updated Blimp input file, run the updated input file by using the `blimp.run()` function, and print the updated Blimp output file by using the `blimp.print()` function.

Usage

```
blimp.update(x, update, file = "Blimp_Input_Update.imp", comment = FALSE,
             replace.inp = TRUE, blimp.run = TRUE, posterior = FALSE,
             folder = "Posterior_",
             format = c("csv", "csv2", "xlsx", "rds", "RData"),
             clear = TRUE, replace.out = c("always", "never", "modified"),
             Blimp = .detect.blimp(),
             result = c("all", "default", "algo.options",  "data.info",
                        "model.info", "warn.mess", "out.model", "gen.param"),
             exclude = NULL, color = c("none", "blue", "violet"),
             style = c("bold", "regular"), not.result = TRUE,
             write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`x`	`misty.object` object of type `blimp`.
`update`	a character vector containing the updated input command sections.

`file` a character string indicating the name of the updated Blimp input file with or without the file extension `.imp`, e.g., `"Blimp_Input_Update.imp"` or `"Blimp_Input_Update.imp"`.

`comment` logical: if `FALSE` (default), comments (i.e., text after the # symbol) are removed from the input text specified in the argument `x`.

`replace.inp` logical: if `TRUE` (default), an existing input file will be replaced.

`blimp.run` logical: if `TRUE`, the input file specified in the argument `file` containing the input text specified in the argument `x` is run using the `blimp.run()` function.

`posterior` logical: if `TRUE`, the posterior distribution including burn-in and post-burn-in phase for all parameters are saved in long format in a file called `posterior.*` in the folder specified in the argument `folder` and `.imp` file name in the format specified in the argument `format`.

`folder` a character string indicating the prefix of the folder for saving the posterior distributions. The default setting is `folder = "Posterior_"`.

`format` a character vector indicating the file format(s) for saving the posterior distributions, i.e., `"csv"` (default) for `write.csv()`, `"csv2"` for `write.csv2()`, `"xlsx"` for `write.xlsx()`, `"rds"` for `saveRDS()`, and `"RData"` for `write()`.

`clear` logical: if `TRUE` (default), the console is cleared after estimating each model.

`replace.out` a character string for specifying three settings: `"always"` (default), which runs all models, regardless of whether an output file for the model exists, `"never"`, which does not run any model that has an existing output file, and `"modified"`, which only runs a model if the modified date for the input file is more recent than the output file modified date.

`Blimp` a character string for specifying the name or path of the Blimp executable to be used for running models. This covers situations where Blimp is not in the system's path, or where one wants to test different versions of the Blimp program. Note that there is no need to specify this argument for most users since it has intelligent defaults.

`result` a character vector specifying Blimp result sections included in the output (see 'Details' in the `blimp.print` function).

`exclude` a character vector specifying Blimp input command or result sections excluded from the output (see 'Details' in the `blimp.print` function).

`color` a character vector with two elements indicating the colors used for headers (e.g., `"ALGORITHMIC OPTIONS SPECIFIED:"`), and for the header `Outcome Variable:` and `Missing predictor:` including variables names.

`style` a character vector with two elements indicating the style used for headers (e.g., `"ALGORITHMIC OPTIONS SPECIFIED:"`), and for the header `Outcome Variable:` and `Missing predictor:` including variables names, i.e., `regular`, for regular text, `bold` for bold text, `italic`, for italic text, and `underline` for underline text.

`not.result` logical: if `TRUE` (default), character vector indicating the result sections not requested are shown on the console.

`write` a character string naming a file for writing the output into a text file with file extension `".txt"` (e.g., `"Output.txt"`).

`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console by using the function `blimp.print()`.
`data`	a matrix or data frame from which the variables names for the section `VARIABLES` are extracted when using the `...` specification in the `VARIABLES` section.

Details

Bimp Input Sections The function is used to update following Blimp input sections:

- `DATA`
- `VARIABLES`
- `CLUSTERID`
- `ORDINAL`
- `NOMINAL`
- `COUNT`
- `WEIGHT`
- `MISSING`
- `LATENT`
- `RANDOMEFFECT`
- `TRANSFORM`
- `BYGROUP`
- `FIXED`
- `CENTER`
- `MODEL`
- `SIMPLE`
- `PARAMETERS`
- `TEST`
- `FCS`
- `SIMUALTE`
- `SEED`
- `BURN`
- `ITERATIONS`
- `CHAINS`
- `NIMPS`
- `THIN`
- `OPTIONS`
- `OUTPUT`
- `SAVE`

The `---;` Specification The `---;` specification is used to remove entire sections (e.g., `CENTER: ---;`) from the Blimp input. Note that `---;` including the semicolon `;` needs to be specified, i.e., `---` without the semicolon `;` will result in an error message.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	`misty.object` object of type `blimp`
`update`	a character vector containing the updated Blimp input command sections
`args`	specification of function arguments
`write`	updated write command sections
`result`	list with result sections (`result`)

Author(s)

Takuya Yanagida

References

Keller, B. T., & Enders, C. K. (2023). *Blimp user's guide* (Version 3). Retrieved from www.appliedmissingdata.com/blimp

See Also

`blimp.run`, `blimp.print`, `blimp.plot`, `blimp.bayes`

Examples

```
## Not run:

#──────────────────────────────────────────────────────────────────────────────
# Example 1a: Update BURN and ITERATIONS section

# Specify Blimp input
input <- '
DATA: data1.csv;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
'

# Run Blimp input
mod0 <- blimp(input, file = "Ex4.3.imp", clear = FALSE)

# Update sections
update1 <- '
BURN: 5000;
```

```
ITERATIONS: 20000;
'

# Run updated Blimp input
mod1 <- blimp.update(mod0, update1, file = "Ex4.3_update1.imp")

#-----------------------------------------------------------------------
# Example 1b: Remove CENTER section

# Remove section
update2 <- '
CENTER: ---;
'

# Run updated Blimp input
mod2 <- blimp.update(mod1, update2, file = "Ex4.3_update2.imp")

## End(Not run)
```

`boot.bs` *Bollen-Stine Bootstrapping with Incomplete Data*

Description

This function performs the model-based Bollen-Stine Bootstrapping with incomplete data of the chi-square statistic. By default, the function performs model-based bootstrapping based on transformation method 2 in Savalei and Yuan (2009).

Usage

```
boot.bs(object = NULL, data = NULL, model = NULL, sigma = NULL, mu = NULL,
        group = NULL, chisq = NULL, em.cov = NULL, trans = c(1, 2), nrep = 500,
        return = c("transdat", "bootsamp", "output"), seed = NULL,
       progress = TRUE, digits = 2, p.digits = 3, plot = FALSE, filename = NULL,
        width = NA, height = NA, dpi = 600, write = NULL, append = TRUE,
        check = TRUE, output = TRUE, ...)
```

Arguments

`object`	an object of class lavaan, i.e., a fitted latent variable model including mean structures, i.e., `meanstructure = TRUE`.
`data`	a data frame representing the target raw data set, optional argument if the argument `object` is not specified. Note that the data frame should only include variables that are used in the covariance matrix and mean vector specified in the arguments `sigma` and `mu`.
`model`	a character string. Optional argument representing the target model if the argument `object` is not specified.

`sigma` a matrix. Optional argument representing the model-implied covariance matrix if the argument `object` is not specified.

`mu` a numeric vector. Optional argument representing the model-implied mean vector if the argument `object` is not specified.

`group` a character vector. Optional argument representing the name of the grouping variable in `data` if the argument `object` is not specified.

`chisq` a numeric value. Optional argument representing the model's χ^2 test statistic if the argument `object` is not specified.

`em.cov` a matrix. Optional argument representing the EM or Two-Stage ML estimated covariance matrix used to speed up the Transformation 2 algorithm.

`trans` a character string representing the transformation method in Savalei and Yuan (2009). There are three methods presented in the article, but only the first two are currently implemented in the function, i.e., `trans = 1` when there are few missing data patterns, each of which has a large size, such as in a planned missing data design, or `trans = 2` (default) when there are more missing data patterns.

`nrep` a numeric value indicating the number of bootstrap replicates (default is 500).

`return` a character string indicating which results to return, i.e., `"transdat"` for only the transformed data, `"bootsamp"` for only the bootstrap samples, or `"output"` (default) for the output table for the Bollen-Stine Bootstrapping of the chi-square statistic.

`seed` a numeric value specifying the seed of the pseudo-random numbers used when drawing bootstrap samples.

`progress` logical: if `TRUE` (default), progress bar will be displayed while fitting the model to the bootstrap samples. Note that a `for` loop is used when `progress = TRUE`, while the `sapply` function is used when `progress = FALSE`.

`digits` an integer value indicating the number of decimal places to be used for displaying the χ^2 test statistic.

`p.digits` an integer value indicating the number of decimal places to be used for displaying the *p*-values.

`plot` logical: if `TRUE`, bootstrap sampling distribution of the χ^2 test statistic is plotted with a histogram including a density curve.

`filename` a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"` (default), `".jpeg"`, `".tiff"`, `".png"`, `".bmp"`, `".svg"` or `".wmf"` needs to be specified as file extension in the `file`

`width` a numeric value indicating the `width` argument (default is the size of the current graphics device) in the `ggsave` function.

`height` a numeric value indicating the `height` argument (default is the size of the current graphics device) in the `ggsave` function.

`dpi` a numeric value indicating the `dpi` argument (default is `600`) in the `ggsave` function.

`write` a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.

`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.
`...`	additional arguments in the lavaan::lavaan() function, see lavaan::lavOptions().

Value

Returns an object of class `misty.object` when specifying `return = "output"`:

`call`	function call
`type`	type of analysis
`object`	object of class lavaan specified in the argument `object`
`args`	specification of function arguments
`plot`	ggplot2 object when specifying `plot = TRUE`
`result`	result table

When specifying `return = "transdat"`, the transformed data and when specifying `return = "bootsamp"`, the bootstrap samples are returned.

Note

This function is based on modified copies of the functions `bsBootMiss` from the **semTools** package by Terrence D. Jorgensen et al. (2026).

Author(s)

Takuya Yanagida

References

Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods & Research, 21*(2), 205-229. https://doi.org/10.1177/0049124192021002004

Jorgensen, T. D., Pornprasertmanit, S., Schoemann, A. M., & Rosseel, Y. (2026). *semTools: Useful tools for structural equation modeling*. R package version 0.5-8. Retrieved from https://CRAN.R-project.org/package=semTools

Savalei, V., & Yuan, K.-H. (2009). On the model-based bootstrap with missing data: Obtaining a p-value for a test of exact fit. *Multivariate Behavioral Research, 44*(6), 741-763. https://doi.org/10.1080/00273170903333590

Examples

```
## Not run:
# Load lavaan package
library(lavaan)

# Holzinger and Swineford data set
```

```
dat <- HolzingerSwineford1939

# Introduce missing data
dat$x5 <- ifelse(dat$x1 <= quantile(dat$x1, 0.3), NA, dat$x5)
dat$x9 <- ifelse(is.na(dat$x5), NA, dat$x9)

# Model specification
model <- 'visual  =~ x1 + x2 + x3
          textual =~ x4 + x5 + x6
          speed   =~ x7 + x8 + x9'

# Model estimation
fit <- sem(model, data = dat, meanstructure = TRUE, std.lv = TRUE,
           missing = "fiml", group = "school")

#-----------------------------------------------------------------------
# Bollen-Stine Bootstrapping with Incomplete Data

# Example 1: Default setting, transformation method 2, 500 replicates
# Plot bootstrap sampling distribution of the test statistic
boot.bs(fit, seed = 42, plot = TRUE)

#-----------------------------------------------------------------------
# Transformed Data and Bootstrap Samples

# Example 2: Return transformed data only
transdat <- boot.bs(fit, return = "transdat")

# Example 3: Return bootstrap samples only
bootsamp <- boot.bs(fit, return = "bootsamp")

#-----------------------------------------------------------------------
# Plot Bootstrap Sampling Distribution of Chi-Square Test Statistic

# Bollen-Stine Bootstrapping
object <- boot.bs(fit, seed = 42)

# Load ggplot2 package
library(ggplot2)

# Plot data
plotdat <- data.frame(chisq = object$boot.chisq)

# Example 3: Plot bootstrap sampling distribution, create plot manually
ggplot(plotdat, aes(chisq)) +
  geom_histogram(aes(y = after_stat(density)), color = "black", alpha = 0.4, fill = "gray85") +
   geom_density(color = "#0072B2") +
   geom_vline(aes(xintercept = object$result$chisq, color = "Observed Test Statistic")) +
   scale_x_continuous(name = expression(paste(chi^2, " Test Statistic")),
                 limits = c(0, max(c(plotdat$chisq, object$result$chisq), na.rm = TRUE))) +
  scale_y_continuous(name = "Probability Density, f(x)", expand = expansion(mult = c(0, 0.05))) +
   scale_color_manual(values = c("Observed Test Statistic" = "#CC79A7")) +
   theme_bw() +
```

```
  theme(legend.position = "bottom", legend.box.margin = margin(-12, 0, 0, 0),
        legend.title = element_blank())

## End(Not run)
```

---

center *Centering Predictor Variables in Single-Level and Multilevel Data*

Description

This function centers predictor variables in single-level data, two-level data, and three-level data at the grand mean (CGM, i.e., grand mean centering) or within clusters (CWC, i.e., group mean centering).

Usage

```
center(data, ..., cluster = NULL, type = c("CGM", "CWC", "latent"),
       cwc.mean = c("L2", "L3"), value = NULL, append = TRUE, name = ".c",
       as.na = NULL, check = TRUE)
```

Arguments

`data`	a numeric vector for centering a predictor variable, or a data frame for centering more than one predictor variable.
`...`	an expression indicating the variable names in `data` e.g., `center(dat, x1, x2)` for centering the variables `x1` and `x2` in the data frame `dat`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`cluster`	a character string indicating the name of the cluster variable in `data` for a two-level model (e.g., `cluster = "level2"`), a character vector indicating the names of the cluster variables in `data` for a three-level model (e.g., `cluster = c("level3", "level2")`), or a vector (e.g., `data$level2`) or data frame (e.g., `data[, c("level3", "level2"]`) representing the nested grouping structure (i.e., group or cluster variables). Note that the cluster variable at Level 3 come first in a three-level model, i.e., `cluster = c("level3", "level2")`.
`type`	a character string indicating the type of centering, i.e., `"CGM"` for centering at the grand mean (i.e., grand mean centering, default when `cluster = NULL`), `"CWC"` for centering within clusters (i.e., group mean centering, default when specifying the argument `cluster`, or `"latent"` for the two-step latent mean centering method (see 'Details'). Note that two-step latent mean centering method can only be applied to one predictor variable at a time.
`cwc.mean`	a character string indicating the type of centering of a Level-1 predictor variable in a three-level model, i.e., `L2` (default) for centering the predictor variable at the Level-2 cluster means, and `L3` for centering the predictor variable at the level-3 cluster means. Note that this argument is only used when specifying two cluster variables for the argument `"cluster"`.

`value` a numeric value for centering on a specific user-defined value. Note that this option is only available when specifying predictor variables in single-level data i.e., `cluster = NULL`.

`append` logical: if `TRUE` (default), centered variable(s) are appended to the data frame specified in the argument `data`.

`name` a character character vector indicating the names of the centered predictor variables. By default, centered predictor variables are named with the ending `".c"` resulting in e.g. `"x1.c"` and `"x2.c"`. Variable names can also be specified by using a character vector matching the number of variables (e.g., `name = c("center.x1", "center.x2")`). Note that when specifying `type = "latent"`, centered predictor variables in a two-level model are named with the endings `".l1"` and `".l2"` (e.g., `name = c("x.l1", "x.l2")`), while centered predictor variables in a three-level model are named with the endings `".l1"`, `".l2"`, and `".l3"` (e.g., `name = c("x.l1", "x.l2", "x.l3")`) by default. Alternatively, a character vector of length 2 for centered predictor variables in a two-level model or a character vector of length 3 centered predictor variables in a three-level model can be specified.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data` but not to `cluster`.

`check` logical: if `TRUE` (default), argument specification is checked.

Details

Single-Level Data

Predictor variables are centered at the grand mean (CGM) by default:

$$x_i - \bar{x}_.$$

where x_i is the predictor value of observation i and $\bar{x}_.$ is the average x score. Note that predictor variables can be centered on any meaningful value specifying the argument `value`, e.g., a predictor variable centered at 5 by applying following formula:

$$x_i - \bar{x}_. + 5$$

resulting in a mean of the centered predictor variable of 5.

Two-Level Data

In two-level data, there are predictor variables at Level-1 (L1) and Level-2 (L2) with L1 predictor variables centered within L2 clusters (CWC) and L2 predictors centered at the average L2 cluster scores (CGM) by default:

- **Level-1 (L1) Predictor Variables**:
  L1 predictor variable can be centered within L2 clusters (CWC) or at the grand-mean (CGM):
  - L1 predictor variables are centered within L2 clusters by specifying `type = "CWC"` (Default):

$$x_{ij} - \bar{x}_{.j}$$

where $\bar{x_{.j}}$ is the average x score in cluster j.

– L1 predictor variables are centered at the grand-mean by specifying `type = "CGM"`:

$$x_{ij} - \bar{x}_{..}$$

where x_{ij} is the predictor value of observation i in L2 cluster j and $\bar{x}_{..}$ is the average x score.

- **Level-2 (L2) Predictor Variables**:
  L2 predictor variables are centered at the average L2 cluster score:

$$x_{.j} - \bar{x}_{..}$$

where $x_{.j}$ is the predictor value of L2 cluster j and $\bar{x}_{..}$ is the average L2 cluster score. Note that the cluster membership variable needs to be specified when centering a L2 predictor variable in two-level data. Otherwise the average x_{ij} individual score instead of the average $x_{.j}$ cluster score is used to center the predictor variable.

Three-Level Data

In three-level data, there are predictor variables at Level-1 (L1), Level-2 (L2), and Level-3 (L3) with L1 predictor variables centered within L2 clusters (CWC L2), L2 predictors centered within L3 clusters (CWC L3), and L3 predictors centered at the average L3 cluster scores (CGM) by default:

- **Level-1 (L1) Predictor Variables**:
  L1 predictor variables can be centered within L2 clusters (CWC L2), within L3 clusters (CWC L3) or at the grand-mean (CGM):

  – L1-predictor variables are centered within cluster (CWC) by specifying `type = "CWC"` (Default). Note that L1 predictor variables can be either centered within L2 clusters (`cwc.mean = "L2"`, Default, see Brincks et al., 2017):

$$x_{ijk} - \bar{x}_{.jk}$$

or within L3 clusters (`cwc.mean = "L3"`, see Enders, 2013):

$$x_{ijk} - \bar{x}_{..k}$$

where $\bar{x}_{.jk}$ is the average x score in L2 cluster j within Level-3 cluster k and $\bar{x}_{..k}$ is the average x score in L3 cluster k.

  – L1 predictor variables are centered at the grand mean (CGM) by specifying `type = "CGM"`:

$$x_{ijk} - \bar{x}_{...}$$

where x_{ijk} is the predictor value of observation i in L2 cluster j within L3 cluster k and $\bar{x}_{...}$ is the average x score.

- **Level-2 (L2) Predictor Variables**:
  L2 predictor variables can be centered within L3 clusters (CWC) or at the L2 grand-mean (CGM):

- L2 predictor variables are centered within cluster by specifying `type = "CWC"` (Default):

$$x_{.jk} - \bar{x}_{..k}$$

  where $\bar{x}_{..k}$ is the average x score in L3 cluster k.
- L2 predictor variables are centered at the grand mean by specifying `type = "CGM"`:

$$x_{.jk} - \bar{x}_{...}$$

  where $x_{.jk}$ is the predictor value of L2 cluster j within L3 cluster k and $\bar{x}_{...}$ is the average L2 cluster score.

- **Level-3 (L3) Predictor Variables**:
  L3-predictor variables are centered at the L3 grand mean:

$$x_{..k} - \bar{x}_{...}$$

  where $x_{..k}$ is the predictor value of L3 cluster k and $\bar{x}_{...}$ is the average L3 cluster score.

Two-Step Latent Mean Centering

The latent mean centering approach (Asparouhov & Muthén, 2019) in a two-level model decomposes the Level-1 predictor variable x_{ij} as within and between compoments as follows:

$$x_{ij} = x_{w,ij} + x_{b,.j}$$

where $x_{w,ij}$ is the individual specific contribution and $x_{b,.j}$ is the cluster specific contribution to the predictor variable x_{ij}. Here, $x_{b,.j}$ can be interpreted as the intercepts and $x_{w,ij}$ can be interpreted as the residuals in the random intercept model. Note that $x_{w,ij}$ is equivalent to a L1 predictor centered within L2 clusters (CWC), while $x_{b,.j}$ is equivalent to a L2 predictor centered at the average L2 cluster scores (CGM). Latent mean centering treats $x_{b,.j}$ as unknown quantity that is estimated while taking into the sampling error in the mean estimate under the assumption of large cluster sizes in the population and less than 5% of the cluster population sampled. As a result, this approach resolves problems that occur with the traditional observed centering methods, e.g., Lüdtke's bias (Lüdtke et al., 2008) in the estimation of contextual effects or Nickell's bias (Asparouhov et al., 2018) in the estimation of the autocorrelations in time-series models.

The latent mean centering approach requires a latent variable modeling program, e.g., commercial software Mplus (Muthen & Muthen, 1998-2017) or the R package lavaan (Rosseel, 2012) and cannot be used in mixed-effects modeling programs like lme4 (Bates et al., 2015) or nlme (Pinheiro & Bates, 2000). In order to mimic the latent mean centering approach, a two-step approach is proposed in the `center()` function, where a random intercept model is fit to the L1 predictor variable to extract the intercepts representing $x_{b,.j}$ and residuals representing $x_{w,ij}$. These two components can be used as L1 predictor centered within clusters and L2 predictor centered at the grand mean. Note that compared to the latent mean centering approach, this two-step approach will result in bias because $x_{w,ij}$ and $x_{b,.j}$ are treated as observed instead of latent variables. However, the magnitude of the bias is unclear without conducting a simulation study. Hence, the latent mean centering using a latent variable modeling program is recommended whenever possible, while the two-step latent mean centering approach implemented in the `center()` function is just an 'experimental' approach that cannot be recommend at this time.

Value

Returns a numeric vector or data frame with the same length or same number of rows as `data` containing the centered variable(s).

Author(s)

Takuya Yanagida

References

Asparouhov, T., Hamaker, E. L., & Muthén, B. (2017). Dynamic Structural Equation Models. *Structural Equation Modeling: A Multidisciplinary Journal, 25*(3), 359-388. https://doi.org/10.1080/10705511.2017.1406803

Asparouhov, T., & Muthén, B. (2019). Latent variable centering of predictors and mediators in multilevel and time-series models. *Structural Equation Modeling, 26*(1), 119-142. https://doi.org/10.1080/10705511.2018.1511375

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software, 67*(1), 1–48. https://doi.org/10.18637/jss.v067.i01

Brincks, A. M., Enders, C. K., Llabre, M. M., Bulotsky-Shearer, R. J., Prado, G., & Feaster, D. J. (2017). Centering predictor variables in three-level contextual models. *Multivariate Behavioral Research, 52*(2), 149–163. https://doi.org/10.1080/00273171.2016.1256753

Chang, C.-N., & Kwok, O.-M. (2022) Partitioning Variance for a Within-Level Predictor in Multilevel Models. *Structural Equation Modeling: A Multidisciplinary Journal*. Advance online publication. https://doi.org/10.1080/10705511.2022.2051175

Enders, C. K. (2013). Centering predictors and contextual effects. In M. A. Scott, J. S. Simonoff, & B. D. Marx (Eds.), *The Sage handbook of multilevel modeling* (pp. 89-109). Sage. https://dx.doi.org/10.4135/9781446247600

Enders, C. K., & Tofighi, D. (2007). Centering predictor variables in cross-sectional multilevel models: A new look at an old issue. *Psychological Methods, 12*, 121-138. https://doi.org/10.1037/1082-989X.12.2.121

Lüdtke, O., Marsh, H. W., Robitzsch, A., Trautwein, U., Asparouhov, T., & Muthén, B. (2008). The multilevel latent covariate model: A new, more reliable approach to group-level effects in contextual studies. *Psychological Methods, 13*(3), 203-229. https://doi.org/10.1037/a0012869

Muthén, L. K., & Muthén, B. O. (1998-2017). *Mplus User's Guide* (8th ed). Muthén & Muthén.

Pinheiro, J. C., & Bates, D. M. (2000). *Mixed-Effects Models in S and S-PLUS*. Springer. https://doi.org/10.1007/b98882

Rights, J. D., Preacher, K. J., & Cole, D. A. (2020). The danger of conflating level-specific effects of control variables when primary interest lies in level-2 effects. *British Journal of Mathematical & Statistical Psychology, 73*, 194-211. https://doi.org/10.1111/bmsp.12194

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*(2), 1-36. https://doi.org/10.18637/jss.v048.i02

Yaremych, H. E., Preacher, K. J., & Hedeker, D. (2021). Centering categorical predictors in multilevel models: Best practices and interpretation. *Psychological Methods*. Advance online publication. https://doi.org/10.1037/met0000434

See Also

coding, cluster.scores, rec, item.reverse, cluster.rwg, item.scores.

Examples

```
#————————————————————————————————————————————————————————————————————
# Single-Level Data

# Example 1a: Center predictor 'disp' at the grand mean
center(mtcars, disp, append = FALSE)

# Example 1b: Alternative specification without using the '...' argument
center(mtcars$disp)

# Example 1c: Center predictors 'disp' and 'hp' at the grand mean and append to 'mtcars'
center(mtcars, disp, hp)

# Example 1d: Alternative specification without using the '...' argument
cbind(mtcars, center(mtcars[, c("disp", "hp")]))

# Example 1e: Center predictor 'disp' at the value 3
center(mtcars, disp, value = 3)

# Example 1f: Center predictors 'disp' and 'hp' and label with the suffix ".v"
center(mtcars, disp, hp, name = ".v")

#————————————————————————————————————————————————————————————————————
## Two-Level Data

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————
## Level-1 (L1) Predictor

# Example 2a: Center L1 predictor 'y1' within L2 clusters
center(Demo.twolevel, y1, cluster = "cluster", append = FALSE)

# Example 2b: Alternative specification without using the '...' argument
center(Demo.twolevel$y1, cluster = Demo.twolevel$cluster)

# Example 2c: Center L1 predictor 'y1' at the grand-mean
#             Note that cluster ID is ignored when type = "CGM"
center(Demo.twolevel, y1, cluster = "cluster", type = "CGM")

# Example 2d: Alternative specification
center(Demo.twolevel, y1)

#————————————————————————————————————
## Level-2 (L2) Predictor

# Example 2e: Center L2 predictor 'w2' at the average L2 cluster scores
```

```
#             Note that cluster ID is needed
center(Demo.twolevel, w1, cluster = "cluster")

#————————————————————————————————————
## L1 and L2 Predictors

# Example 2f: Center L1 predictor 'y1' within L2 clusters
#             and L2 predictor 'w1' at the average L2 cluster scores
center(Demo.twolevel, y1, w1, cluster = "cluster")

#————————————————————————————————————
## Twp-Step Latent Mean Centering

# Example 2g: Decompose L1 predictor 'y1' as within-between components
center(Demo.twolevel, y1, cluster = "cluster", type = "latent")

# Example 2h: Decompose L1 predictor 'y1' as within-between components
#             label variables as 'l1.y1' and 'l2.y1'
center(Demo.twolevel, y1, cluster = "cluster", type = "latent", name = c("l1.y1", "l2.y1"))

## Not run:
#————————————————————————————————————————————————————————————————————————
# Three-Level Data

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

# Compute L3 cluster scores for the L2 predictor 'w1'
Demo.threelevel <- cluster.scores(Demo.threelevel, w1, cluster = "cluster3", name = "w1.l3")

#————————————————————————————————————
## Level-1 (L1) Predictor

# Example 3a: Center L1 predictor 'y1' within L2 clusters (CWC L2)
#             Note that L3 cluster IDs are ignored when type = "CWC"
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"))

# Example 3b: Alternative specification when L2 cluster IDs are unique across L3 clusters
center(Demo.threelevel, y1, cluster = "cluster2")

# Example 3c: Center L1 predictor 'y1' within L3 clusters (CWC L3)
#             Note that both L3 and L2 cluster IDs are needed
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), cwc.mean = "L3")

# Example 3d: Center L1 predictor 'y1' at the grand-mean (CGM)
#             Note that the cluster argument is ignored when type = "CGM",
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "CGM")

# Example 3e:  Alternative specification
```

```
center(Demo.threelevel, y1)

#————————————————————————————————————
## Level-2 (L2) Predictor

# Example 3f: Center L2 predictor 'w1' within L3 cluster
#             Note that both L3 and L2 cluster IDs are needed
center(Demo.threelevel, w1, cluster = c("cluster3", "cluster2"))

# Example 3g: Center L2 predictor 'w1' at the grand-mean (CGM)
#             Note that both L3 and L2 cluster IDs are needed
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "CGM")

#————————————————————————————————————
## Level-3 (L3) Predictor

# Example 3h: Center L3 predictor 'w1.l3' at the average L3 cluster scores
#             Note that L2 cluster ID is ignored
center(Demo.threelevel, w1.l3, cluster = c("cluster3", "cluster2"))

# Example 3i: Alternative specification
center(Demo.threelevel, w1.l3, cluster = "cluster3")

#————————————————————————————————————
## L1, L2, and L3 Predictors

# Example 3j: Center L1 predictor 'y1' within L2 cluster, L2 predictor 'w1' within
#             L3 clusters, and L3 predictor 'w1.l3' at the average L3 cluster scores
center(Demo.threelevel, y1, w1, w1.l3, cluster = c("cluster3", "cluster2"))

#————————————————————————————————————
## Two-Step Latent Mean Centering

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

# Example 3k: Decompose L1 predictor 'y1' as within-between components
center(Demo.threelevel, y1, cluster = "cluster2", type = "latent")

# Example 3l: Decompose L1 predictor 'y1' as within-between components
#             label variables as 'l1.y1' and 'l2.y1'
center(Demo.threelevel, y1, cluster = "cluster2", type = "latent",
       name = c("l1.y1", "l2.y2"))

# Example 3m: Decompose L1 predictor 'y1' as within-between components
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "latent")

# Example 3n: Decompose L1 predictor 'y1' as within-between components
#             label variables as 'l1.y1', 'l2.y1', and 'l3.y1'
```

```
center(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "latent",
       name = c("l1.y1", "l2.y1", "l3.y1"))

## End(Not run)
```

`check.collin` *Collinearity Diagnostics*

Description

This function computes tolerance, standard error inflation factor, variance inflation factor, eigenvalues, condition index, and variance proportions for linear, generalized linear, and mixed-effects models.

Usage

```
check.collin(model, print = c("all", "vif", "eigen"), digits = 3, p.digits = 3,
             write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted model of class `"lm"`, `"glm"`, `"lmerMod"`, `"lmerModLmerTest"`, `"glmerMod"`, `"lme"`, or `"glmmTMB"`.
`print`	a character vector indicating which results to show, i.e. `"all"`, for all results, `"vif"` for tolerance, std. error inflation factor, and variance inflation factor, or `eigen` for eigenvalue, condition index, and variance proportions.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`write`	a character string naming a text file with file extension `".txt"` (e.g., `"Output.txt"`) for writing the output into a text file.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Collinearity diagnostics can be conducted for objects returned from the `lm()` and `glm()` function, but also from objects returned from the `lmer()` and `glmer()` function from the **lme4** package, `lme()` function from the **nlme** package, and the `glmmTMB()` function from the **glmmTMB** package.

The generalized variance inflation factor (Fox & Monette, 1992) is computed for terms with more than 1 df resulting from factors with more than two levels. The generalized VIF (GVIF) is interpretable as the inflation in size of the confidence ellipse or ellipsoid for the coefficients of the term

in comparison with what would be obtained for orthogonal data. GVIF is invariant to the coding of the terms in the model. In order to adjust for the dimension of the confidence ellipsoid, $\text{GVIF}^{\frac{1}{2df}}$ is computed. Note that the adjusted GVIF (aGVIF) is actually a generalized standard error inflation factor (GSIF). Thus, the aGIF needs to be squared before applying a common cutoff threshold for the VIF (e.g., VIF > 10). Note that the output of `check.collin()` function reports either the variance inflation factor or the squared generalized variance inflation factor in the column `VIF`, while the standard error inflation factor or the adjusted generalized variance inflation factor is reported in the column `SIF`.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in the `model` argument
`args`	specification of function arguments
`result`	list with result tables, i.e., `coef` for the regression table including tolerance, std. error inflation factor and variance inflation factors, `vif` for the tolerance, std. error inflation factor, and variance inflation factor, and `eigen` for eigenvalue condition index, and variance proportion

Note

The computation of the VIF and the GVIF is based on the `vif()` function in the **car** package by John Fox, Sanford Weisberg and Brad Price (2020), and the computation of eigenvalues, condition index, and variance proportions is based on the `ols_eigen_cindex()` function in the **olsrr** package by Aravind Hebbali (2020).

Author(s)

Takuya Yanagida

References

Fox, J., & Monette, G. (1992). Generalized collinearity diagnostics. *Journal of the American Statistical Association, 87*, 178-183.

Fox, J., Weisberg, S., & Price, B. (2020). *car: Companion to Applied Regression*. R package version 3.0-8. https://cran.r-project.org/web/packages/car/

Hebbali, A. (2020). *olsrr: Tools for building OLS regression models*. R package version 0.5.3. https://cran.r-project.org/web/packages/olsrr/

See Also

`check.outlier`, `lm`

Examples

```
dat <- data.frame(group = c(1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4),
                  x1 = c(3, 2, 4, 9, 5, 3, 6, 4, 5, 6, 3, 5),
                  x2 = c(1, 4, 3, 1, 2, 4, 3, 5, 1, 7, 8, 7),
                  x3 = c(7, 3, 4, 2, 5, 6, 4, 2, 3, 5, 2, 8),
                  x4 = c("a", "b", "a", "c", "c", "c", "a", "b", "b", "c", "a", "c"),
                  y1 = c(2, 7, 4, 4, 7, 8, 4, 2, 5, 1, 3, 8),
                  y2 = c(0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1), stringsAsFactors = TRUE)

#————————————————————————————————————————————————————————————————————————
# Linear model

# Estimate linear model with continuous predictors
mod.lm1 <- lm(y1 ~ x1 + x2 + x3, data = dat)

# Example 1: Tolerance, std. error, and variance inflation factor
check.collin(mod.lm1)

# Example 2: Tolerance, std. error, and variance inflation factor
# Eigenvalue, Condition index, and variance proportions
check.collin(mod.lm1, print = "all")

# Estimate model with continuous and categorical predictors
mod.lm2 <- lm(y1 ~ x1 + x2 + x3 + x4, data = dat)

# Example 3: Tolerance, generalized std. error, and variance inflation factor
check.collin(mod.lm2)

#————————————————————————————————————————————————————————————————————————
# Generalized linear model

# Estimate logistic regression model with continuous predictors
mod.glm <- glm(y2 ~ x1 + x2 + x3, data = dat, family = "binomial")

# Example 4: Tolerance, std. error, and variance inflation factor
check.collin(mod.glm)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Linear mixed-effects model

# Load lme4, nlme, and glmmTMB package
libraries(lme4, nlme, glmmTMB)

# Estimate linear mixed-effects model using lme4 package
mod.lmer <- lmer(y1 ~ x1 + x2 + x3 + (1|group), data = dat)

# Example 5: Tolerance, std. error, and variance inflation factor
check.collin(mod.lmer)

# Estimate linear mixed-effects model using nlme package
mod.lme <- lme(y1 ~ x1 + x2 + x3, random = ~ 1 | group, data = dat)
```

```
# Example 6: Tolerance, std. error, and variance inflation factor
check.collin(mod.lme)

# Estimate linear mixed-effects model using glmmTMB package
mod.glmmTMB1 <- glmmTMB(y1 ~ x1 + x2 + x3 + (1|group), data = dat)

# Example 7: Tolerance, std. error, and variance inflation factor
check.collin(mod.glmmTMB1)

#————————————————————————————————————————————————————————————————————
# Generalized linear mixed-effects model

# Estimate mixed-effects logistic regression model using lme4 package
mod.glmer <- glmer(y2 ~ x1 + x2 + x3 + (1|group), data = dat, family = "binomial")

# Example 8: Tolerance, std. error, and variance inflation factor
check.collin(mod.glmer)

# Estimate mixed-effects logistic regression model using glmmTMB package
mod.glmmTMB2 <- glmmTMB(y2 ~ x1 + x2 + x3 + (1|group), data = dat, family = "binomial")

# Example 9: Tolerance, std. error, and variance inflation factor
check.collin(mod.glmmTMB2)

#————————————————————————————————————————————————————————————————————
# Write Results

# Example 10: Write Results into a text file
check.collin(mod.lm1, write = "Diagnostics.txt")

## End(Not run)
```

`check.outlier` *Outlier and Influential Observations Diagnostics for Linear Models*

Description

This function performs outlier and influential observations diagnostics by plotting for statistical measures for leverage, distance, and influence for linear models estimated by using the `lm()` function. *Mahalanobis distance* and *hat values* are computed for quantifying **leverage**, *standardized leverage-corrected residuals* and *studentized leverage-corrected residuals* are computed for quantifying **distance**, and *Cook's distance* and standardized *DFBETAS* are computed for quantifying **influence**. By default, the function provides plots showing observations with the 10% highest absolute values or a minimum of 10 observations with the highest absolute values in the *hat values*, *studentized leverage-corrected residuals*, and *Cook's distances*.

Usage

```
check.outlier(model, type = c("mahal", "hat", "rstand", "rstud", "cook", "dfb"),
```

```
cutoff = NULL, sample = 0.10, sort = TRUE, append = TRUE,
plot = TRUE, geom = c("none", "text"), filename = NULL,
width = NA, height = NA, dpi = 600, check = TRUE, ...)
```

Arguments

model	a fitted model of class "lm".
type	a character vector indicating which statistical measures to include in the plot, i.e., "mahal" for the Mahalanobis distance, "hat" for the hat value, "rstand" for the standardized leverage-corrected residual, "rstud" for the studentized leverage-corrected residual, "cook" for the Cook's distance, and "dfb" for the DFBETAS.
cutoff	a named numeric vector indicating the cutoff values for the statistical measures shown in the plot as horizontal lines. Note that the elements of the vectors need to be need with "mahal", "hat", "rstand", "rstud", "cook", and "dfb". For example, cutoff = c(hat = 0.2, cook = 0.3) to draw a horizontal line at 0.2 for the hat value plot and a horizontal line at 0.3 for the Cook's distance plot.
sample	numeric value for selecting the sample with the highest absolute values in the statistical measures for leverage, distance, and influence. The value is either a proportion between 0 and 1, e.g., sample = 0.2 for the 20% highest absolute values, or an integer value for selecting the number of observations with the highest absolute values. Note that the logical argument FALSE can be used to plot all observations. By default, the function plots observations with the 10% highest absolute values, but a minimum of 10 observations.
sort	logical: if TRUE (default), observations are sorted in decreasing order by the statistical measure.
append	logical: logical: if TRUE (default), statistical measures for leverage, distance, and influence are appended to the data frame in model$model.
plot	logical: if TRUE (default), a plot is drawn.
geom	a character vector indicating which plot elements to draw, i.e., "none" for not drawing any labels, "text" (default) for drawing labels above or below each observation.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
check	logical: if TRUE (default), argument specification is checked.
...	further arguments to be passed to or from methods.

Details

In regression analysis, an observation can be extreme in three major ways (see Darlington & Hayes, 2017, p. 484): (1) An observation has high **leverage** if it has a atypical pattern of values on the predictors, (2) an observation has high **distance** if its observed outcome value Y_i has a large deviation from the predicted value $\hat{Y}_i$, and (3) an observation has high **influence** if its inclusion substantially changes the estimates for the intercept and/or slopes.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`args`	specification of function arguments
`plot`	ggplot2 object for plotting the residuals
`data`	list with data frame, i.e., `data` for a data frame in `model$model` with statistical measures for leverage, distance, and influence are appended to the data frame when `append = TRUE`, `mahal` for the sample of `data` with the highest Mahalanobis distance, `hat` for the sample of `data` with the highest hat values, `rstand` for the sample of `data` with the highest absolute standardized leverage-corrected residuals, `rstud` for the sample of `data` with the highest absolute studentized leverage-corrected residuals, `cook` for the sample of `data` with the highest Cook's distance, and `*.dfb` for the sample of `data` with the highest absolute DFBETAS

Author(s)

Takuya Yanagida

References

Darlington, R. B., &, Hayes, A. F. (2017). *Regression analysis and linear models*: Concepts, applications, and implementation. The Guilford Press.

See Also

`check.resid`, `check.collin`

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————————
# Outlier and Influential Observations Diagnostics for Linear Models

# Estimate linear model
mod <- lm(mpg ~ cyl + disp + hp, data = mtcars)
```

```
#————————————————————————————————————
## Default Setting

# Example 1a: Hat values, studentized leverage-corrected residuals, and Cook's distance
check.outlier(mod)

# Example 1b: Additional plot arguments
# see Details in the help page of the function plot.misty.object
check.outlier(mod, point.shape = 1, point.size = 4, line.col = "red")

#————————————————————————————————————
## Argument 'cutoff' and 'type'

# Example 2a: Horizontal cutoff lines
check.outlier(mod, cutoff = c(hat = 0.3, rstud = 2, cook = 0.1))

# Example 2b: Mahalanobis distance and standardized leverage-corrected residuals
check.outlier(mod, type = c("mahal", "rstand"), cutoff = c(mahal = 8, rstand = 2))

# Example 2c: DFBETAS
check.outlier(mod, type = "dfb", cutoff = c(dfb = 0.35))

#————————————————————————————————————
## Arguments 'sample' and 'sort'

# Example 3a: Plot entire data set and do not sort
check.outlier(mod, sample = FALSE, sort = FALSE)

# Example 3b: Plot 50% highest absolute values
check.outlier(mod, sample = 0.5)

# Example 3c: Plot n = 15 highest absolute values
check.outlier(mod, sample = 15)

#————————————————————————————————————
## Data

object <- check.outlier(mod)

# Example 4a: Data frame with with statistical measures
object$data$data

# Example 4b: Sample of data with the highest hat values
object$data$hat

#————————————————————————————————————————————————————————————————————
# Save Plot

# Example 7: Save plot as PNG fine
check.outlier(mod, filename = "Outlier.png", width = 12, height = 15)

## End(Not run)
```

check.resid *Residual Diagnostics for Linear, Multilevel and Mixed-Effects Models*

Description

This function performs residual diagnostics for linear models estimated by using the `lm()` function and for multilevel and linear mixed-effects models estimated by using the `lmer()` function from the **lme4** package to detect nonlinearity (partial residual or component-plus-residual plots), nonconstant error variance (predicted values vs. residuals plot), and non-normality of residuals (Q-Q plot and histogram with density curves).

Usage

```
check.resid(model, type = c("linear", "homo", "normal"),
            resid = c("unstand", "stand", "student"), plot = TRUE,
            geom = c("all", "none", "lm", "loess", "ci.lm", "ci.loess",
                     "hline", "qq", "hist"),
          filename = NULL, width = NA, height = NA, dpi = 600, check = TRUE, ...)
```

Arguments

`model` a fitted model of class `"lm"`, `"lmerMod"`, or `"lmerModLmerTest"`.

`type` a character string specifying the type of the plot, i.e., `"linear"` for partial (component-plus-residual) plots, `"homo"` (default) for predicted values vs. residuals plot, and `"normal"` for Q-Q plot and histogram with a density plot. Note that partial residual or component-plus-residual plots are not available for models with interaction terms

`resid` a character string specifying the type of residual used for the partial (component-plus-residual) plots or Q-Q plot and histogram, i.e., `"unstand"` for unstandardized residuals `"stand"` for standardized residuals, and `"student"` for studentized residuals. By default, studentized residuals are used for predicted values vs. residuals plot and unstandardized residuals are used for Q-Q plot and histogram. Note that studentized residuals are not available for multilevel and linear mixed-effects models when requesting Q-Q plots and histograms.

`plot` logical: if `TRUE` (default), a plot is drawn.

`geom` a character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"none"` for drawing no regression or smoothing curve, `"lm"` for drawing the regression line, `"loess"` for drawing the LOESS smoothing curve, `"ci.lm"` for drawing confidence bands around the regression line, `"ci.loess"` for drawing confidence bands around the LOESS smoothing curve, `"hline"` or drawing a horizontal line at `0`, `"qq"` for plotting Q-Q plots, and `"hist"` for plotting histograms. Note that the arguments `"lm"`, `"loess"`, `"ci.lm"`, and `"ci.less"` are applicable when `type = "linear"`, the arguments `"hline"`, `"loess"`, and `"ci.loess"` are applicable when `type = "homo"`, and the arguments `"lqq"` and `"hist"` are applicable when `type = "norm"`.

filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
check	logical: if TRUE (default), argument specification is checked.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

Nonlinearity The violation of the assumption of linearity implies that the model cannot accurately capture the systematic pattern of the relationship between the outcome and predictor variables. In other words, the specified regression surface does not accurately represent the relationship between the conditional mean values of Y and the Xs. That means the average error $E(\varepsilon)$ is not 0 at every point on the regression surface (Fox, 2015).

In multiple regression, plotting the outcome variable Y against each predictor variable X can be misleading because it does not reflect the partial relationship between Y and X (i.e., statistically controlling for the other Xs), but rather the marginal relationship between Y and X (i.e., ignoring the other Xs). Partial residual plots or component-plus-residual plots should be used to detect nonlinearity in multiple regression. The partial residual for the jth predictor variable is defined as

$$e_i^{(j)} = b_j X_{ij} + e_i$$

The linear component of the partial relationship between Y and X_j is added back to the least-squares residuals, which may include an unmodeled nonlinear component. Then, the partial residual $e_i^{(j)}$ is plotted against the predictor variable X_j. Nonlinearity may become apparent when a non-parametric regression smoother is applied.

By default, the function plots each predictor against the partial residuals, and draws the linear regression and the LOESS smoothing curve to the partial residual plots.

Nonconstant Error Variance The violation of the assumption of constant error variance, often referred to as heteroscedasticity, implies that the variance of the outcome variable around the regression surface is not the same at every point on the regression surface (Fox, 2015).

Plotting residuals against the outcome variable Y instead of the predicted values $\hat{Y}$ is not recommended because $Y = \hat{Y} + e$. Consequently, the linear correlation between the outcome variable Y and the residuals e is $\sqrt{1 - R^2}$ where R is the multiple correlation coefficient. In contrast, plotting residuals against the predicted values $\hat{Y}$ is much easier to examine for evidence of nonconstant error variance as the correlation between $\hat{Y}$ and e is 0. Note that the least-squares residuals generally have unequal variance $Var(e_i) = \sigma^2/(1 - h_i)$ where h is the leverage of observation i, even if errors have constant variance σ^2. The studentized

residuals e_i^*, however, have a constant variance under the assumption of the regression model. Residuals are studentized by dividing them by $\sigma_i^2(\sqrt{(1-h_i)}$ where σ_i^2 is the estimate of σ^2 obtained after deleting the ith observation, and h_i is the leverage of observation i (Meuleman et al, 2015).

By default, the function plots the predicted values against the studentized residuals. It also draws a horizontal line at 0, a LOESS smoothing curve for all residuals as well as separate LOESS smoothing curves for positive and negative residuals.

Non-normality of Residuals Statistical inference under the violation of the assumption of normally distributed errors is approximately valid in all but small samples. However, the efficiency of least squares is not robust because the least-squares estimator is the most efficient and unbiased estimator only when the errors are normally distributed. For instance, when error distributions have heavy tails, the least-squares estimator becomes much less efficient compared to robust estimators. In addition, error distributions with heavy-tails result in outliers and compromise the interpretation of conditional means because the mean is not an accurate measure of central tendency in a highly skewed distribution. Moreover, a multimodal error distribution suggests the omission of one or more discrete explanatory variables that naturally divide the data into groups (Fox, 2016).

By default, the function plots a Q-Q plot of the unstandardized residuals, and a histogram of the unstandardized residuals and a density plot. Note that studentized residuals follow a t-distribution with $n-k-2$ degrees of freedom where n is the sample size and k is the number of predictors. However, the normal and t-distribution are nearly identical unless the sample size is small. Moreover, even if the model is correct, the studentized residuals are not an independent random sample from t_{n-k-2}. Residuals are correlated with each other depending on the configuration of the predictor values. The correlation is generally negligible unless the sample size is small.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`args`	specification of function arguments
`plot`	ggplot2 object for plotting the residuals

Note

This function uses a modified copy of the `partial()` and `calc_ranef()` function in the **remef** package by Sven Hohenstein and Reinhold Kliegl (2025) when requesting partial residual plots for linear mixed-effects models.

Author(s)

Takuya Yanagida

References

Fox, J. (2016). *Applied regression analysis and generalized linear models* (3rd ed.). Sage Publications, Inc.

Hohenstein, S., & Kliegl, R. (2025). *remef: Remove Partial Effects*. R package version 1.0.7, https://github.com/hohenstein/remef

Meuleman, B., Loosveldt, G., & Emonds, V. (2015). Regression analysis: Assumptions and diagnostics. In H. Best & C. Wolf (Eds.), *The SAGE handbook of regression analysis and causal inference (pp. 83-110)*. Sage.

See Also

check.collin, check.outlier

Examples

```
#————————————————————————————————————————————————————————————————————————
# Linear Model

# Estimate linear model
mod.lm <- lm(Ozone ~ Solar.R + Wind + Temp, data = airquality)

#————————————————————————————————————
## Nonlinearity

# Example 1a: Partial (component-plus-residual) plots
check.resid(mod.lm, type = "linear")

# Example 1b: Plot with smoothing curve only
check.resid(mod.lm, type = "linear", geom = "loess")

# Example 1c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lm, type = "linear", point.size = 1, point.shape = 2)

#————————————————————————————————————
## Nonconstant Error Variance

# Example 2a: Predicted values vs. residuals plot
check.resid(mod.lm, type = "homo")

# Example 2b: Plot with horizontal line only
check.resid(mod.lm, type = "homo", geom = "hline")

# Example 2c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lm, type = "homo", point.shape = 2, linetype = "solid", line.col = "blue")

#————————————————————————————————————
## Non-Normality of Residuals

# Example 3a: Q-Q plot and histogram with density plot
```

```
check.resid(mod.lm, type = "normal")

# Example 3b: Q-Q plot only
check.resid(mod.lm, type = "normal", geom = "qq")

# Example 2c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lm, type = "normal", point.shape = 2, abline.col = "red")

## Not run:
#------------------------------------------------------------------------
# Multilevel and Linear Mixed-Effects Model

# Load lme4 package
library(lme4)

# Estimate two-level mixed-effects model
mod.lmer <- lmer(Reaction ~ Days + (Days | Subject), data = sleepstudy)

#-------------------------------------
## Nonlinearity

# Example 4a: Partial (component-plus-residual) plots
check.resid(mod.lmer, type = "linear")

# Example 4b: Plot with smoothing curve only
check.resid(mod.lmer, type = "linear", geom = "loess")

# Example 4c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lmer, type = "linear", point.size = 1, point.shape = 2)

#-------------------------------------
## Nonconstant Error Variance

# Example 5a: Predicted values vs. residuals plot
check.resid(mod.lmer, type = "homo")

# Example 5b: Plot with horizontal line only
check.resid(mod.lmer, type = "homo", geom = "hline")

# Example 5c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lmer, type = "homo", point.shape = 2, linetype = "solid",
            line.col = "blue")

#-------------------------------------
## Non-Normality of Residuals

# Example 6c: Q-Q plot and histogram with density plot
check.resid(mod.lmer, type = "normal")

# Example 6b: Q-Q plot only
```

```
check.resid(mod.lmer, type = "normal", geom = "qq")

# Example 6c: Additional arguments
# see Details in the help page of the function plot.misty.object
check.resid(mod.lmer, type = "normal", point.shape = 2, abline.col = "red")

#-----------------------------------------------------------------------
# Save Plot

# Example 7: Save plot as PNG fine
check.resid(mod.lm, type = "linear", filename = "Linear.png", width = 15, height = 5)

## End(Not run)
```

---

chr.color *Colored and Styled Terminal Output Text*

Description

This function adds color and style to output texts on terminals that support 'ANSI' color and highlight codes that can be printed by using the `cat` function.

Usage

```
chr.color(x, color = c("black", "red", "green", "yellow", "blue", "violet",
                       "cyan", "white", "gray1", "gray2", "gray3",
                       "b.red", "b.green", "b.yellow", "b.blue", "b.violet",
                       "b.cyan", "b.white"),
          bg = c("none", "black", "red", "green", "yellow", "blue", "violet",
                 "cyan", "white"),
          style = c("regular", "bold", "italic", "underline"), check = TRUE)
```

Arguments

`x`	a character vector.
`color`	a character string indicating the text color, e.g., `"red"` for red and `b."red"` for bright red text.
`bg`	a character string indicating the background color of the text, e.g., `"red"` for red background.
`style`	a character vector indicating the font style, i.e., `"regular"` (default) for regular text, `"bold"` for bold text, `"italic"`, for italic text, and `"underline"` for underline text. Note that font styles can be combined, e.g., `style = c("bold", "italic")` provides a bold and italic text.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns a character vector.

Note

This function is based on functions provided in the **crayon** package by Gábor Csárdi.

Author(s)

Takuya Yanagida

References

Csárdi G (2022). *crayon: Colored Terminal Output*. R package version 1.5.2, https://CRAN.R-project.org/package=crayon

See Also

chr.grep, chr.grepl, chr.gsub, chr.omit, chr.trim, chr.trunc

Examples

```
## Not run:

# Example 1:
cat(chr.color("Text in red.", color = "red"))

# Example 2:
cat(chr.color("Text in blue with green background.",
              color = "blue", bg = "yellow"))

# Example 3a:
cat(chr.color("Text in boldface.", style = "bold"))

# Example 3b:
cat(chr.color("Text in boldface and italic.", style = c("bold", "italic")))


## End(Not run)
```

---

chr.grep *Multiple Pattern Matching*

Description

This function searches for matches to the character vector specified in `pattern` within each element of the character vector `x`.

Usage

```
chr.grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
         fixed = FALSE, useBytes = FALSE, invert = FALSE, check = TRUE)

chr.grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE,
          useBytes = FALSE, check = TRUE)
```

Arguments

`pattern`	a character vector with character strings to be matched.
`x`	a character vector where matches are sought.
`ignore.case`	logical: if `FALSE` (default), the pattern matching is case sensitive and if `TRUE`, case is ignored during matching.
`perl`	logical: if `TRUE` Perl-compatible regexps are used.
`value`	logical: if `FALSE` (default), a vector containing the (integer) indices of the matches determined by grep is returned, and if `TRUE`, a vector containing the matching elements themselves is returned.
`fixed`	logical: if `TRUE`, pattern is a string to be matched as is. Overrides all conflicting arguments.
`useBytes`	logical: if `TRUE`, the matching is done byte-by-byte rather than character-by-character. See 'Details'.
`invert`	logical: if `TRUE`, function returns indices or values for elements that do not match.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns a integer vector with the indices of the mathces when `value = FALSE`, character vector containing the matching elements when `value = TRUE`, or a logical vector when using the `chr.grepl` function.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole

See Also

`chr.color`, `chr.grepl`, `chr.gsub`, `chr.omit`, `chr.trim`, `chr.trunc`

Examples

```
chr.vector <- c("James", "Mary", "Michael", "Patricia", "Robert", "Jennifer")

# Example 1: Indices of matching elements
chr.grep(c("am", "er"), chr.vector)

# Example 2: Values of matching elements
chr.grep(c("am", "er"), chr.vector, value = TRUE)

# Example 3: Matching element?
chr.grepl(c("am", "er"), chr.vector)
```

`chr.gsub` *Multiple Pattern Matching And Replacements*

Description

This function is a multiple global string replacement wrapper that allows access to multiple methods of specifying matches and replacements.

Usage

```
chr.gsub(pattern, replacement, x, recycle = FALSE, check = TRUE, ...)
```

Arguments

`pattern`	a character vector with character strings to be matched.
`replacement`	a character vector equal in length to `pattern` or of length one which are a replacement for matched patterns.
`x`	a character vector where matches and replacements are sought.
`recycle`	logical: if `TRUE`, replacement is recycled if lengths differ.
`check`	logical: if `TRUE` (default), argument specification is checked.
`...`	additional arguments to pass to the `regexpr` or `sub` function.

Value

Return a character vector of the same length and with the same attributes as `x` (after possible coercion to character).

Note

This function was adapted from the `mgsub()` function in the **mgsub** package by Mark Ewing (2019).

Author(s)

Takuya Yanagida

References

Mark Ewing (2019). *mgsub: Safe, Multiple, Simultaneous String Substitution*. R package version 1.7.1. https://CRAN.R-project.org/package=mgsub

See Also

chr.color, chr.grep, chr.grepl, chr.omit, chr.trim, chr.trunc

Examples

```
# Example 1: Replace 'the' and 'they' with 'a' and 'we'
chr.vector <- "they don't understand the value of what they seek."
chr.gsub(c("the", "they"), c("a", "we"), chr.vector)

# Example 2: Replace 'heyy' and 'ho' with 'yo'
chr.vector <- c("hey ho, let's go!")
chr.gsub(c("hey", "ho"), "yo", chr.vector, recycle = TRUE)

# Example 3: Replace with regular expressions
chr.vector <- "Dopazamine is not the same as dopachloride or dopastriamine, yet is still fake."
chr.gsub(c("[Dd]opa([^ ]*?mine)","fake"), c("Meta\1","real"), chr.vector)
```

---

chr.omit *Omit Strings*

Description

This function omits user-specified values or strings from a numeric vector, character vector or factor.

Usage

```
chr.omit(x, omit = "", na.omit = FALSE, check = TRUE)
```

Arguments

x	a numeric vector, character vector or factor.
omit	a numeric vector or character vector indicating values or strings to be omitted from the vector x, the default setting is the empty strings "".
na.omit	logical: if TRUE, missing values (NA) are also omitted from the vector.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a numeric vector, character vector or factor with values or strings specified in omit omitted from the vector specified in x.

Author(s)

Takuya Yanagida

See Also

chr.color, chr.grep, chr.grepl, chr.gsub, chr.trim, chr.trunc

Examples

```
#-----------------------------------------------------------------------
# Charater Vector

x.chr <- c("a", "", "c", NA, "", "d", "e", NA)

# Example 1: Omit character string ""
chr.omit(x.chr)

# Example 2: Omit character string "" and missing values (NA)
chr.omit(x.chr, na.omit = TRUE)

# Example 3: Omit character string "c" and "e"
chr.omit(x.chr, omit = c("c", "e"))

# Example 4: Omit character string "c", "e", and missing values (NA)
chr.omit(x.chr, omit = c("c", "e"), na.omit = TRUE)

#-----------------------------------------------------------------------
# Numeric Vector

x.num <- c(1, 2, NA, 3, 4, 5, NA)

# Example 5: Omit values 2 and 4
chr.omit(x.num, omit = c(2, 4))

# Example 6: Omit values 2, 4, and missing values (NA)
chr.omit(x.num, omit = c(2, 4), na.omit = TRUE)

#-----------------------------------------------------------------------
# Factor

x.factor <- factor(letters[1:10])

# Example 7: Omit factor levels "a", "c", "e", and "g"
chr.omit(x.factor, omit = c("a", "c", "e", "g"))
```

---

chr.trim *Trim Whitespace from String*

Description

This function removes whitespace from start and/or end of a string

Usage

```
chr.trim(x, side = c("both", "left", "right"), check = TRUE)
```

Arguments

x	a character vector.
side	a character string indicating the side on which to remove whitespace, i.e., "both" (default), "left" or "right".
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a character vector with whitespaces removed from the vector specified in x.

Note

This function is based on the str_trim() function from the **stringr** package by Hadley Wickham.

Author(s)

Takuya Yanagida

References

Wickham, H. (2019). *stringr: Simple, consistent wrappers for common string operations*. R package version 1.4.0.

See Also

chr.color, chr.grep, chr.grepl, chr.gsub, chr.omit, chr.trunc

Examples

```
x <- "  string  "

# Example 1: Remove whitespace at both sides
chr.trim(x)

# Example 2: Remove whitespace at the left side
chr.trim(x, side = "left")

# Example 3: Remove whitespace at the right side
chr.trim(x, side = "right")
```

chr.trunc *Truncate a Character Vector to a Maximum Width*

Description

This function truncates a character vector, so that the number of characters of each element of the character vector is always less than or equal to the width specified in the argument `width`.

Usage

```
chr.trunc(x, width, side = c("right", "left", "center"), ellipsis = "...",
          check = TRUE)
```

Arguments

`x`	a character vector or factor. Note that factors are converted into a character vector.
`width`	a numeric value indicating the maximum width of the character strings in the vector. Note that the default setting switches to `"..."` when `width = 3`, `"."` when `width = 2`, and `""` when `width = 1`.
`side`	a character string indicating the location of the ellipsis, i.e. `"right"` (default) for the right side, `"left"` for the left side, and `"center"` for center of the character strings in the vector
`ellipsis`	a character string indicating the content of the ellipsis, i.e., `"..."` by default.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns a truncated character vector.

Note

This function was adapted from the `str_trunc()` function in the **stringr** package by Hadley Wickham (2023).

Author(s)

Takuya Yanagida

References

Wickham H (2023). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.5.1, https://CRAN.R-project.org/package=stringr

See Also

`chr.color`, `chr.grep`, `chr.grepl`, `chr.gsub`, `chr.omit`, `chr.trim`

Examples

```
# Example 1: Truncate at the right side with a max. of 10 characters
chr.trunc(row.names(mtcars), width = 10)

# Example 2: Truncate at the left side with a max. of 10 characters
chr.trunc(row.names(mtcars), width = 10, side = "left")

# Example 3: Truncate without ellipses
chr.trunc(row.names(mtcars), width = 10, ellipsis = "")
```

`ci.cor` *(Bootstrap) Confidence Intervals for Correlation Coefficients*

Description

This function computes and plots (1) Fisher z' confidence intervals for Pearson product-moment correlation coefficients (a) without non-normality adjustment, (1b) adjusted via sample joint moments method or (1c) adjusted via approximate distribution method (Bishara et al., 2018), (2) Spearman's rank-order correlation coefficients with (2a) Fieller et al. (1957) standard error, (2b) Bonett and Wright (2000) standard error, or (2c) rank-based inverse normal transformation, (3) Kendall's Tau-b, and (4) Kendall-Stuart's Tau-c correlation coefficients with Fieller et al. (1957) standard error, optionally by a grouping and/or split variable. The function also supports five types of bootstrap confidence intervals (e.g., bias-corrected (BC) percentile bootstrap or bias-corrected and accelerated (BCa) bootstrap confidence intervals) and plots the bootstrap samples with histograms and density curves. By default, the function computes Pearson product-moment correlation coefficients adjusted via approximate distribution method.

Usage

```
ci.cor(data, ...,
       method = c("pearson", "spearman", "kendall-b", "kendall-c"),
       adjust = c("none", "joint", "approx"),
       se = c("fisher", "fieller", "bonett", "rin"),
       sample = TRUE, seed = NULL, maxtol = 1e-05, nudge = 0.001,
       boot = c("none", "norm", "basic", "perc", "bc", "bca"), nrep = 1000,
       fisher = TRUE, alternative = c("two.sided", "less", "greater"),
       conf.level = 0.95, group = NULL, split = NULL, na.omit = FALSE,
       digits = 2, as.na = NULL, plot = c("none", "ci", "boot"),
       geom = c("all", "hist", "density", "point", "ci", "line"),
       filename = NULL, width = NA, height = NA, dpi = 600,
       write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with numeric variables, i.e., factors and character variables are excluded from `data` before conducting the analysis.

`...` an expression indicating the variable names in `data` e.g., `ci.cor(x1, x2, data = dat)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`method` a character string indicating which correlation coefficient is to be computed, i.e., `"pearson"` for Pearson product-moment correlation coefficient (default), `"spearman"` for Spearman's rank-order correlation coefficient, `"kendall-b"` for Kendall's Tau-b correlation coefficient, `"kendall-c"` for Kendall-Stuart's Tau-c correlation coefficient. Note that confidence intervals are only computed given at least 4 pairs of observations.

`adjust` a character string specifying the non-normality adjustment method, i.e., `"none"` for the Fisher z' confidence interval for the Pearson product-moment correlation coefficient without non-normality adjustment, `"joint"` for the confidence interval with non-normality adjustment via sample joint moments, and `"approx"` (default) for the confidence interval with non-normality adjustment via approximate distribution by skewness and kurtosis. Note that this argument only applies to the Pearson product-moment correlation coefficient, i.e., `method = "pearson"`

`se` a character string specifying the method for computing the standard error of the correlation coefficient, i.e., `"fisher"` for the Fisher z' confidence interval, `"fieller"` (default) for the confidence interval for Spearman's rank-order correlation coefficient based on approximate standard error by Fieller et al. (1957), `"bonett"` for the confidence interval based on approximate standard error by Bonett and Wright (2000), and `"rin"` for the confidence interval for Spearman's rank-order correlation coefficient based on rank-based inverse normal (RIN) transformation. Note that this argument only applies to Spearman's rank-order correlation coefficient, i.e., `method = "spearman"`.

`sample` logical: if `TRUE` (default), the univariate sample skewness and kurtosis is used when applying the approximate distribution method and reported in the result table, while the population skewness and kurtosis is used when `sample = FALSE`.

`seed` a numeric value specifying the seed of the pseudo-random number generator when generating a random set of starting parameter value when the parameters led to a sum of squares greater than the maximum tolerance after optimization when applying the approximate distribution method (`adjust = approx`) when computing the confidence interval for the Pearson product-moment correlation coefficient, or the seeds of the pseudo-random numbers used when conducting bootstrapping.

`maxtol` a numeric value indicating the tolerance for total squared error when applying the approximate distribution method (`adjust = approx`).

`nudge` a numeric value indicating the nudge proportion of their original values by which sample skewness, kurtosis, and r are nudged towards 0 when applying the approximate distribution method (`adjust = approx`). are only computed given at least 10 pairs of observations.

`boot` a character string specifying the type of bootstrap confidence intervals (CI), i.e., `"none"` (default) for not conducting bootstrapping, `"norm"` for the bias-corrected normal approximation bootstrap CI, `"basic"` for the basic bootstrap CI, `"perc"`, for the percentile bootstrap CI `"bc"` for the bias-corrected (BC) percentile bootstrap CI (without acceleration), and `"bca"` for the bias-corrected and accelerated (BCa) bootstrap CI.

nrep	a numeric value indicating the number of bootstrap replicates (default is 1000).
fisher	logical: if TRUE (default), Fisher z transformation is applied before computing the confidence intervals to reverse-transformed the limits of the interval using the inverse of the Fisher z transformation. Note that this argument applies only when boot is "norm" or "basic".
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	either a character string indicating the variable name of the grouping variable in ... or data, or a vector representing the grouping variable. The grouping variable is excluded from the data frame specified in data.
split	either a character string indicating the variable name of the split variable in ... or data, or a vector representing the split variable. The split variable is excluded from the data frame specified in data.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion); if FALSE (default), pairwise deletion is used.
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	a character string indicating the type of the plot to display, i.e., "none" (default) for not displaying any plots, "ci" for displaying confidence intervals for the correlation coefficient, "boot" for displaying bootstrap samples with histograms and density curves when the argument "boot" is other than "none".
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "hist" for drawing histograms when plotting bootstrap samples (plot = "boot"), "density" for drawing density curves when plotting bootstrap samples (plot = "boot"), "point" for drawing vertical lines representing the point estimate of the correlation coefficients when plotting bootstrap samples (plot = "boot"), "ci" for drawing vertical lines representing the bootstrap confidence intervals of the correlation when plotting bootstrap samples (plot = "boot"), and "line" for drawing a horizontal line when specifying plot = "ci" or a vertical line when specifying plot = "boot".
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument. Note that plots can only be saved when plot = "ci" or plot = "boot".
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.

`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Pearson Product-Moment Correlation Coefficient The Fisher z' confidence interval method for the Pearson product-moment correlation coefficient is based on the assumption that X and Y have a bivariate normal distribution in the population. Non-normality resulting from either high kurtosis or high absolute skewness can distort the Fisher z' confidence interval that produces a coverage rate that does not equal the one intended. The distortion is largest when population correlation is large and both variables X and Y were non-normal (Bishara et al., 2017). Note that increasing sample size improves coverage only when the population correlation is zero, while increasing the sample size worsens coverage with a non-zero population correlation (Bishara & Hittner, 2017). The `ci.cor` function computes the Fisher z' confidence interval without non-normality adjustment (`adjust = "none"`), with non-normality adjustment via sample joint moments (`adjust = "joint"`), or with non-normality adjustment via approximate distribution (`adjust = "approx"`):

- *Fisher z' confidence interval method* uses the r-toz' transformation for the correlation coefficient r:

  $$z' = 0.5 \cdot \ln\left(\frac{1+r}{1-r}\right)$$

  The sampling distribution of z is approximately normal with a standard error of approximately

  $$\sigma'_z = \sqrt{\frac{1}{(n-3)}}$$

  The two-sided 95% confidence interval is defined as

  $$z' \pm 1.96 \cdot \sigma_{z'}$$

  These confidence interval bounds are transformed back to the scale of r:

  $$r = \frac{exp(2z') - 1}{exp(2z') + 1}$$

  The resulting confidence interval of the correlation coefficient is an approximation and is only accurate when X and Y have a bivariate normal distribution in the population or when the population correlation is zero.
- The *Joint Moments Method* multiplies the asymptotic variance of z' by τ_f^2 (Hawkins, 1989):

$$\tau_f^2 = \frac{(\mu_{40} + 2\mu_{22} + \mu_{04})\rho^2 - 4(\mu_{31} + \mu_{13})\rho + 4\mu_{22})}{4(1-\rho^2)^2}$$

where μ_{jk} represents a population joint moment defined as

$$\mu_{jk} = E[X^j Y^k]$$

where X and Y are assumed to be standardized ($\mu_{10} = \mu_{01} = 0$, $\mu_{20} = \mu_{02} = 1$). The standard error of z' can then be approximated as $\tilde{\sigma}_{z'}$:

$$\tilde{\sigma}_{z'} = \tau_f \sqrt{\frac{1}{n-3}}$$

The corresponding sample moments, m_{jk} can be used to estimate τ_f^2:

$$\hat{\mu}_{jk} = m_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_i^j y_i^k)$$

However, the higher-order sample joint moments may be unstable estimators of their population counterparts unless the sample size is extremely large. Thus, this estimate of τ_f^2 may be inaccurate, leading to inaccurate confidence intervals.

- The *Approximate Distribution Method* estimates an approximate distribution that the sample appears to be drawn from to analytically solve for τ_f^2 based on that distribution's parameters. The `ci.cor` function uses a third-order polynomial family allowing estimation of distribution parameters using marginal skewness and kurtosis that are estimated using the marginal sample skewness and kurtosis statistics (Bishara et al., 2018).

Bishara et al. (2018) conducted two Monte Carlo simulations that showed that the approximate distribution method was effective in dealing with violations of the bivariate normality assumption for a wide range of sample sizes, while the joint moments method was effective mainly when the sample size was extremely large, in the thousands. However, the third-order polynomial family used for the approximate distribution method cannot deal with absolute skewness above 4.4 or kurtosis above 43.4. Note that the approximate distribution method is accurate even when the bivariate normality assumption is satisfied, while the sample joint moments method sometimes fails to achieve the intended coverage even when the bivariate normality was satisfied.

Spearman's Rank-Order Correlation Coefficient The confidence interval for Spearman's rank-order correlation coefficient is based on the Fisher's z method (`se = "fisher"`), Fieller et al. (1957) approximate standard error (`se = "fieller"`, default), Bonett and Wright (2000) approximate standard error (`se = "bonett"`) or rank-based inverse normal (RIN) transformation (`se = "rin"`):

- *Fisher's z Standard Error*

$$\sqrt{\frac{1}{(n-3)}}$$

- *Fieller et al. (1957) Approximate Standard Error*

$$\sqrt{\frac{1.06}{(n-3)}}$$

Note that this approximation for the standard error is recommended for $n > 10$ and $|rs| < 0.8$.

- *Bonett and Wright (2000) Approximate Standard Error*

$$\sqrt{\frac{1+\frac{\hat{\theta}^2}{2}}{(n-3)}}$$

where $\hat{\theta}$ is the point estimate of the Spearman's rank-order correlation coefficient. Note that this approximation for the standard error is recommended for $|\tau| \leq 0.9$.

- *Rin Transformation* involves three steps. First, the variable is converted to ranks. Second, the ranks are converted to a 0-to-1 scale using a linear function. Third, this distribution is transformed via the inverse of the normal cumulative distribution function (i.e., via probit transformation). The result is an approximately normal distribution regardless of the original shape of the data, so long as ties are infrequent and n is not too small.

Kendall's Tau-b and Tau-c Correlation Coefficient The confidence interval for Kendall's Tau-b and Tau-c correlation coefficient is based on the approximate standard error by Fieller et al. (1957):

$$\sigma'_z = \sqrt{\frac{0.437}{(n-4)}}$$

Note that this approximation for the standard error is recommended for $n > 10$ and $|\tau| < 0.8$.

Bootstrap Confidence Intervals The `ci.cor` function supports bootstrap confidence intervals (CI) for the correlation coefficient by changing the default setting `boot = "none"` to request one of five different types of bootstrap CI (see Efron & Tibshirani, 1993; Davidson & Hinkley, 1997):

- `"norm"`: The bias-corrected normal approximation bootstrap CI relies on the normal distribution based on the standard deviation of the bootstrap samples $\hat{SE}^*$. The function corrects for the bootstrap bias, i.e., difference between the bootstrap estimate $\hat{\theta}^*$ and the sample statistic $\hat{\theta}$ centering the interval at $2\hat{\theta} - \hat{\theta}^*$. The BC normal CI of intended coverage of $1 - 2(\alpha/2)$ is given by

$$Normal : (\hat{\theta}_{low}, \hat{\theta}_{upp} = \hat{\theta} - (\hat{\theta}^* - \hat{\theta}) + z^{\alpha/2} \cdot \hat{SE}^*, \hat{\theta} - (\hat{\theta}^* - \hat{\theta}) + z^{1-\alpha/2} \cdot \hat{SE}^*$$

where $z^{\alpha/2}$ and $z^{1-\alpha/2}$ denotes the α and the $1-\alpha$ quantile from the standard normal distribution.

- `"basic"`: The basic bootstrap (aka reverse bootstrap percentile) CI is based on the distribution of $\hat{\delta} = \hat{\theta} - \theta$ which is approximated with the bootstrap distribution of $\hat{\delta}^* = \hat{\theta}^* - \hat{\theta}$.

$$Basic : (\hat{\theta}_{low}, \hat{\theta}_{upp} = \hat{\theta} - \hat{\delta}^{*1-(\alpha/2)}, \hat{\theta} - \hat{\delta}^{*\alpha/2} = 2\hat{\theta} - \hat{\theta}^{*(1-\alpha/2)}, 2\hat{\theta} - \hat{\theta}^{*(\alpha/2)})$$

- `"perc"`: The percentile bootstrap CI is computed by ordering the bootstrap estimates $\hat{\theta}^*_1, \ldots, \hat{\theta}^*_B$ to determine the $(100(\alpha)/2)$th and $(100(1-\alpha)/2)$th empirical percentile with intended coverage of $1 - 2(\alpha/2)$:

$$Percentile : (\hat{\theta}_{low}, \hat{\theta}_{upp} = \hat{\theta}^{*(1-\alpha/2)}, \hat{\theta}^{*(\alpha/2)})$$

- "bc" (default): The bias-corrected (BC) percentile bootstrap CI corrects the percentile bootstrap CI for median bias of $\hat{\theta}^*$, i.e., the discrepancy between the median of $\hat{\theta}^*$ and $\hat{\theta}$ in normal units. The bias correction $\hat{z}_0$ is obtained from the proportion of bootstrap replications less than the sample estimate $\hat{\theta}$:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#\hat{\theta}_b^* < \hat{\theta}}{B}\right)$$

where $\Phi^{-1}(.)$ represents the inverse function of the standard normal cumulative distribution function and B is the number of bootstrap replications. The BC percentile CI of intended coverage of $1 - 2(\alpha/2)$ is given by

$$BC : (\hat{\theta}_{low}, \hat{\theta}_{upp} = \hat{\theta}^{*(\alpha_1)}, \hat{\theta}^{*(\alpha_2)})$$

where

$$\alpha_1 = \Phi(2\hat{z}_0 + z^{\alpha/2})$$

$$\alpha_2 = \Phi(2\hat{z}_0 + z^{1-\alpha/2})$$

where $\Phi(.)$ represents the standard normal cumulative distribution function and $z^{\alpha/2}$ is the $100(\alpha/2)$ percentile of a standard normal distribution.

- "bca": The bias-corrected and accelerated (BCa) bootstrap CI corrects the percentile bootstrap CI for median bias $\hat{z}_0$ and for acceleration or skewness $\hat{a}$, i.e., the rate of change of the standard error of $\hat{\theta}$ with respect to the true parameter value θ on a normalized scale. The standard normal approximation $\hat{\theta} \sim N(\theta, SE^2)$ assumes that the standard error of $\hat{\theta}$ is the same for all θ. The acceleration constant $\hat{a}$ corrects for this unrealistic assumption and can be computed by using jackknife resampling:

$$\hat{a} = \frac{\sum_{i=1}^{n}(\hat{\theta}_{(.)} - \hat{\theta}_{(i)})^3}{6\{\sum_{i=1}^{n}(\theta_{(.)} - \hat{\theta}_{(i)})^2\}^{3/2}}$$

where $\hat{\theta}_{(i)}$ is the sample estimate with the ith observation deleted and $\hat{\theta}_{(.)} = \sum_{i=1}^{n}\frac{\hat{\theta}_{(i)}}{n}$. Note that the function uses infinitesimal jackknife instead of regular leave-one-out jackknife that down-weights each observation by an infinitesimal amount of $\frac{0.001}{n}$ instead of removing observations. The BCa percentile CI of intended coverage of $1 - 2(\alpha/2)$ is given by

$$BCa : (\hat{\theta}_{low}, \hat{\theta}_{upp} = \hat{\theta}^{*(\alpha_1)}, \hat{\theta}^{*(\alpha_2)})$$

where

$$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z^{\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z^{\alpha/2})}\right)$$

$$\alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z^{1-\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z^{1-\alpha/2})}\right)$$

Note that Fisher transformation is applied before computing the confidence intervals to reverse-transform the limits of the interval using the inverse of the Fisher z transformation (`fisher = TRUE`) when specifying `"norm"` or `"basic"` for the argument `boot`. In addition, interpolation on the normal quantile scale is applied for `"basic"`, `"perc"`, `"bc"`, and `"bca"` when a non-integer order statistic is required (see equation 5.8 in Davison & Hinkley, 1997).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the input specified in `...`, `data`, `group`, and `split`
`args`	specification of function arguments
`boot`	data frame with bootstrap replicates of the correlation coefficient when bootstrapping was requested
`plot`	ggplot2 object for plotting the results
`result`	result table

Note

This function is based on a modified copy of the functions provided in the supporting information in Bishara et al. (2018) for the sample joint moments method and approximate distribution method, functions provided in the supplementary materials in Bishara and Hittner (2017) for Fieller et al. (1957) and Bonett and Wright (2000) correction, and a function provided by Thom Baguley (2024) for the rank-based inverse normal (RIN) transformation. Bootstrap confidence intervals are computed using the R package `boot` by Angelo Canty and Brain Ripley (2024).

Author(s)

Takuya Yanagida

References

Baguley, T. (2024). *CI for Spearman's rho*. seriousstats. https://rpubs.com/seriousstats/616206

Bonett, D. G., & Wright, T. A. (2000). Sample size requirements for estimating Pearson, Kendall and Spearman correlations. *Psychometrika, 65*, 23-28. https://doi.org/10.1007/BF02294183

Bishara, A. J., & Hittner, J. B. (2012). Testing the significance of a correlation with nonnormal data: Comparison of Pearson, Spearman, transformation, and resampling approaches. *Psychological Methods, 17*(3), 399–417. https://doi.org/10.1037/a0028087

Bishara, A. J., & Hittner, J.B. (2017). Confidence intervals for correlations when data are not normal. *Behavior Research Methods, 49*, 294-309. https://doi.org/10.3758/s13428-016-0702-8

Bishara, A. J., Li, J., & Nash, T. (2018). Asymptotic confidence intervals for the Pearson correlation via skewness and kurtosis. *British Journal of Mathematical and Statistical Psychology, 71*(1), 167–185. https://doi.org/10.1111/bmsp.12113

Brown, M. B., & Benedetti, J. K. (1977). Sampling behavior of tests for correlation in two-way contingency tables. *Journal of the American Statistical Association, 72* (358), 309-315. https://doi.org/10.1080/01621459.1977.10480995

Canty, A., & Ripley, B. (2024). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-31.

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap methods and their application*. Cambridge University Press.

Efron, B., & Tibshirani, R. (1993). *An introduction to the bootstrap*. Chapman & Hall.

Fieller, E. C., Hartley, H. O., & Pearson, E. S. (1957). Tests for rank correlation coefficients: I. *Biometrika, 44*, 470-481. https://doi.org/10.2307/2332878

Fisher, R. A. (1921). On the "Probable Error" of a Coefficient of Correlation Deduced from a Small Sample. *Metron*, 1, 3-32.

Hawkins, D. L. (1989). Using U statistics to derive the asymptotic distribution of Fisher's Z statistic. *American Statistician, 43*, 235–237. https://doi.org/10.2307/2685369

Hollander, M., Wolfe, D. A., & Chicken, E. (2015). *Nonparametric statistical methods*. Wiley.

See Also

cor.matrix, ci.mean, ci.mean.diff, ci.prop, ci.var, ci.sd

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————
# Pearson Product-Moment Correlation Coefficient

# Example 1a: Approximate distribution method
ci.cor(mtcars, mpg, drat, qsec)

# Alternative specification without using the '...' argument
ci.cor(mtcars[, c("mpg", "drat", "qsec")])

# Example 1b: Joint moments method
ci.cor(mtcars, mpg, drat, qsec, adjust = "joint")

#————————————————————————————————————————————————————————————————————
# Spearman's Rank-Order Correlation Coefficient

# Example 2a: Fieller et al. (1957) approximate standard error
ci.cor(mtcars, mpg, drat, qsec, method = "spearman")

# Example 2b: Bonett and Wright (2000) approximate standard error
ci.cor(mtcars, mpg, drat, qsec, method = "spearman", se = "bonett")

# Example 2c: Rank-based inverse normal (RIN) transformation
ci.cor(mtcars, mpg, drat, qsec, method = "spearman", se = "rin")

#————————————————————————————————————————————————————————————————————
# Kendall's Tau
```

```
# Example 3a:  Kendall's Tau-b
ci.cor(mtcars, mpg, drat, qsec, method = "kendall-b")

# Example 3b:  Kendall's Tau-c
ci.cor(mtcars, mpg, drat, qsec, method = "kendall-c")

#-----------------------------------------------------------------------
# Grouping Variable

# Example 4a: Grouping variable
ci.cor(mtcars, mpg, drat, qsec, group = "vs")

# Example 4b:  Alternative specification without using the argument '...'
ci.cor(mtcars[, c("mpg", "drat", "qsec")], group = mtcars$vs)

#-----------------------------------------------------------------------
# Split Variable

# Example 5a: Split variable
ci.cor(mtcars, mpg, drat, qsec, split = "am")

# Example 5b:  Alternative specification without using the argument '...'
ci.cor(mtcars[, c("mpg", "drat", "qsec")], split = mtcars$am)

#-----------------------------------------------------------------------
# Grouping and Split Variable

# Example 6a: Grouping and split variable
ci.cor(mtcars, mpg, drat, qsec, group = "vs", split = "am")

# Example 6b:  Alternative specification without using the argument '...'
ci.cor(mtcars[, c("mpg", "drat", "qsec")], group = mtcars$vs, split = mtcars$am)

#-----------------------------------------------------------------------
# Bootstrap Confidence Interval (CI)

# Example 7a: Bias-corrected (BC) percentile bootstrap CI
ci.cor(mtcars, mpg, drat, qsec, boot = "bc")

# Example 7b: Bias-corrected and accelerated (BCa) bootstrap CI,
# 5000 bootstrap replications, set seed of the pseudo-random number generator
ci.cor(mtcars, mpg, drat, qsec, boot = "bca", nrep = 5000, seed = 42)

#-----------------------------------------------------------------------
# Plot Confidence Intervals

# Example 8a: Pearson product-moment correlation coefficient
ci.cor(mtcars, mpg, drat, qsec, plot = "ci")

# Example 8b: Grouping variable
ci.cor(mtcars, mpg, drat, qsec, group = "vs", plot = "ci")

# Example 8c: Split variable
```

```
ci.cor(mtcars, mpg, drat, qsec, split = "am", plot = "ci")

# Example 8d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.cor(mtcars, mpg, drat, qsec, plot = "ci")
plot(object, ybreaks = seq(-1, 1, by = 0.25), title = "Confidence Intervals")

#-----------------------------------------------------------------------
# Plot Bootstrap Samples

# Example 9a: Pearson product-moment correlation coefficient
ci.cor(mtcars, mpg, drat, qsec, boot = "bc", plot = "boot")

# Example 9b: Grouping variable
ci.cor(mtcars, mpg, drat, qsec, group = "vs", boot = "bc", plot = "boot")

# Example 9c: Split variable
ci.cor(mtcars, mpg, drat, qsec, split = "am", boot = "bc", plot = "boot")

# Example 9d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.cor(mtcars, mpg, drat, qsec, boot = "bc", plot = "boot")
plot(object, fill = "gray30", title = "Bootstrap Samples")

#-----------------------------------------------------------------------
# Write Results and Save Plot

# Example 10a: Write results into a text file
ci.cor(mtcars, mpg, drat, qsec, write = "CI_Cor_Text.txt")

# Example 10b: Write results into an Excel file
ci.cor(mtcars, mpg, drat, qsec, write = "CI_Cor_Excel.xlsx")

# Example 10c: Save plot as PNG file
ci.cor(mtcars, mpg, drat, qsec, plot = "ci", filename = "CI_Cor.png",
       width = 8, height = 6)

## End(Not run)
```

`ci.mean` *(Bootstrap) Confidence Intervals for Arithmetic Means and Medians*

Description

The function `ci.mean` computes and plots confidence intervals for arithmetic means with known or unknown population standard deviation or population variance and the function `ci.median` computes confidence intervals for medians, optionally by a grouping and/or split variable. These functions also supports six types of bootstrap confidence intervals (e.g., bias-corrected (BC) percentile bootstrap or bias-corrected and accelerated (BCa) bootstrap confidence intervals) and plots the bootstrap samples with histograms and density curves.

Usage

```
ci.mean(data, ..., sigma = NULL, sigma2 = NULL, adjust = FALSE,
        boot = c("none", "norm", "basic", "stud", "perc", "bc", "bca"),
        nrep = 1000, seed = NULL, sample = TRUE,
        alternative = c("two.sided", "less", "greater"),
        conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
       na.omit = FALSE, digits = 2, as.na = NULL, plot = c("none", "ci", "boot"),
        geom = c("all", "hist", "density", "point", "ci", "line"),
        filename = NULL, width = NA, height = NA, dpi = 600,
        write = NULL, append = TRUE, check = TRUE, output = TRUE)

ci.median(data, ..., boot = c("none", "norm", "basic", "stud", "perc", "bc", "bca"),
          nrep = 1000, seed = NULL, sample = TRUE,
          alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
        na.omit = FALSE, digits = 2, as.na = NULL, plot = c("none", "ci", "boot"),
          geom = c("all", "hist", "density", "point", "ci", "line"),
          filename = NULL, width = NA, height = NA, dpi = 600,
          write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data` a numeric vector or data frame with numeric variables, i.e., factors and character variables are excluded from `data` before conducting the analysis.

`...` an expression indicating the variable names in `data` e.g., `ci.mean(x1, x2, data = dat)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`sigma` a numeric vector indicating the population standard deviation when computing confidence intervals for the arithmetic mean with known standard deviation Note that either argument `sigma` or argument `sigma2` is specified and it is only possible to specify one value for the argument `sigma` even though multiple variables are specified in `data`.

`sigma2` a numeric vector indicating the population variance when computing confidence intervals for the arithmetic mean with known variance. Note that either argument `sigma` or argument `sigma2` is specified and it is only possible to specify one value for the argument `sigma2` even though multiple variables are specified in `data`.

`adjust` logical: if `TRUE`, difference-adjustment for the confidence intervals for the arithmetic mean (Baguley, 2012) is applied.

`boot` a character string specifying the type of bootstrap confidence intervals (CI), i.e., `"none"` (default) for not conducting bootstrapping, `"norm"` for the bias-corrected normal approximation bootstrap CI, `"basic"` for the basic bootstrap CI, `"stud"` for the studentized bootstrap CI, `"perc"`, for the percentile bootstrap CI `"bc"` for the bias-corrected (BC) percentile bootstrap CI (without acceleration), and `"bca"` for the bias-corrected and accelerated (BCa) bootstrap CI, see 'Details' in the `ci.cor` function.

`nrep` a numeric value indicating the number of bootstrap replicates (default is 1000).

`seed` a numeric value specifying seeds of the pseudo-random numbers used in the bootstrap algorithm when conducting bootstrapping.

`sample` logical: if `TRUE` (default), the univariate sample skewness and kurtosis is computed, while the population skewness and kurtosis is computed when `sample = FALSE`.

`alternative` a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.

`conf.level` a numeric value between 0 and 1 indicating the confidence level of the interval.

`group` either a character string indicating the variable name of the grouping variable in `data`, or a vector representing the grouping variable. The grouping variable is excluded from the data frame specified in `data`. Notethat a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.

`split` either a character string indicating the variable name of the split variable in `data`, or a vector representing the split variable. The split variable is excluded from the data frame specified in `data`.Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.

`sort.var` logical: if `TRUE`, output table is sorted by variables when specifying `group`.

`na.omit` logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.

`digits` an integer value indicating the number of decimal places to be used.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data`, but not to `group` or `split`.

`plot` a character string indicating the type of the plot to display, i.e., `"none"` (default) for not displaying any plots, `"ci"` for displaying confidence intervals for the arithmetic mean or median, `"boot"` for displaying bootstrap samples with histograms and density curves when the argument `"boot"` is other than `"none"`.

`geom` a character string or character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"hist"` for drawing histograms when plotting bootstrap samples (`plot = "boot"`), `"density"` for drawing density curves when plotting bootstrap samples (`plot = "boot"`), `"point"` for drawing vertical lines representing the point estimate of the arithmetic mean or median when plotting bootstrap samples (`plot = "boot"`), `"ci"` for drawing vertical lines representing the bootstrap confidence intervals of the arithmetic mean or median when plotting bootstrap samples (`plot = "boot"`), and `"line"` for drawing a horizontal line when specifying `plot = "ci"` or a vertical line when specifying `plot = "boot"`.

`filename` a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"` (default), `".jpeg"`, `".tiff"`, `".png"`, `".bmp"`, `".svg"` or `".wmf"` needs to be specified as file extension in the `file` argument. Note that plots can only be saved when `plot = "ci"` or `plot = "boot"`.

width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	list with the input specified in data, group, and split
args	specification of function arguments
boot	data frame with bootstrap replicates of the arithmetic mean of median when bootstrapping was requested
plot	ggplot2 object for plotting the results and the data frame used for plotting
result	result table

Note

Bootstrap confidence intervals are computed using the R package boot by Angelo Canty and Brain Ripley (2024).

Author(s)

Takuya Yanagida

References

Baguley, T. S. (2012). *Serious stats: A guide to advanced statistics for the behavioral sciences*. Palgrave Macmillan.

Canty, A., & Ripley, B. (2024). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-31.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

test.z, test.t, ci.mean.diff, ci.cor, ci.prop, ci.var, ci.sd, descript

Examples

```
#------------------------------------------------------------------------
# Confidence Interval (CI) for the Arithmetic Mean

# Example 1a: Two-Sided 95% CI
ci.mean(mtcars)

# Example 1b: Two-Sided 95% Difference-Adjusted CI
ci.mean(mtcars, adjust = TRUE)

# Example 1c: Two-Sided 95% CI with known population standard deviation
ci.mean(mtcars, mpg, sigma = 6)

# Alternative specification without using the '...' argument
ci.mean(mtcars$mpg, sigma = 6)

#------------------------------------------------------------------------
# Confidence Interval (CI) for the Median

# Example 2a: Two-Sided 95% CI
ci.median(mtcars)

# Example 2b: One-Sided 99% CI
ci.median(mtcars, alternative = "less", conf.level = 0.99)

#------------------------------------------------------------------------
# Grouping Variable

# Example 3a: Grouping variable
ci.mean(mtcars, mpg, cyl, disp, group = "vs")

# Example 3b: Alternative specification without using the '...' argument
ci.mean(mtcars[, c("mpg", "cyl", "disp")], group = mtcars$vs)

# Example 3c: Sort result table by variables
ci.mean(mtcars, mpg, cyl, disp, group = "vs", sort.var = TRUE)

#------------------------------------------------------------------------
# Split Variable

# Example 4a: Split variable
ci.mean(mtcars, mpg, cyl, disp, split = "am")

# Example 4b:  Alternative specification
ci.mean(mtcars[, c("mpg", "cyl", "disp")], split = mtcars$am)

#------------------------------------------------------------------------
# Grouping and Split Variable
```

```
# Example 5a: Grouping and split variable
ci.mean(mtcars, mpg, cyl, disp, group = "vs", split = "am")

# Example 5b: Alternative specification
ci.mean(mtcars[, c("mpg", "cyl", "disp")], group = mtcars$vs, split = mtcars$am)

# Example 5c: Sort result table by variables
ci.mean(mtcars, mpg, cyl, disp, group = "vs", split = "am", sort.var = TRUE)

## Not run:
#———————————————————————————————————————————————————————————————————————
# Bootstrap Confidence Interval (CI)

# Example 6a: Bias-corrected (BC) percentile bootstrap CI
ci.mean(mtcars, boot = "bc")

# Example 6b: Bias-corrected and accelerated (BCa) bootstrap CI,
# 5000 bootstrap replications, set seed of the pseudo-random number generator
ci.mean(mtcars, boot = "bca", nrep = 5000, seed = 42)

#———————————————————————————————————————————————————————————————————————
# Plot Confidence Intervals

# Example 7a: Two-Sided 95
ci.mean(mtcars, disp, hp, plot = "ci")

# Example 7b: Grouping variable
ci.mean(mtcars, disp, hp, group = "vs", plot = "ci")

# Example 7c: Split variable
ci.mean(mtcars, disp, hp, split = "am", plot = "ci")

# Example 7d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.mean(mtcars, disp, hp, plot = "ci")
plot(object, ybreaks = seq(0, 300, by = 50), title = "Confidence Intervals")

#———————————————————————————————————————————————————————————————————————
# Plot Bootstrap Samples

# Example 8a: Two-Sided 95
ci.mean(mtcars, disp, hp, boot = "bc", plot = "boot")

# Example 8b: Grouping variable
ci.mean(mtcars, disp, hp, group = "vs", boot = "bc", plot = "boot")

# Example 8c: Split variable
ci.mean(mtcars, disp, hp, split = "am", boot = "bc", plot = "boot")

# Example 8d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.mean(mtcars, disp, hp, boot = "bc", plot = "boot")
```

```
plot(object, fill = "gray30", title = "Bootstrap Samples")

#═══════════════════════════════════════════════════════════════════════
# Write Results and Save Plot

# Example 9a: Write results into a text file
ci.mean(mtcars, write = "CI_Mean_Text.txt")

# Example 9b: Write results into an Excel file
ci.mean(mtcars, write = "CI_Mean_Excel.xlsx")

# Example 9c: Save plot as PNG file
ci.mean(mtcars, disp, hp, plot = "ci", filename = "CI_Mean.png",
        width = 9, height = 6)

## End(Not run)
```

`ci.mean.diff` *Confidence Interval for the Difference in Arithmetic Means*

Description

This function computes confidence intervals for the difference in arithmetic means in a one-sample, two-sample and paired-sample design with known or unknown population standard deviation or population variance for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.mean.diff(...)

## Default S3 method:
ci.mean.diff(data = NULL, ..., x = NULL, y, mu = 0,
             sigma = NULL, sigma2 = NULL, var.equal = FALSE, paired = FALSE,
             alternative = c("two.sided", "less", "greater"),
             conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
             digits = 2, as.na = NULL, write = NULL, append = TRUE,
             check = TRUE, output = TRUE)

## S3 method for class 'formula'
ci.mean.diff(formula, data, sigma = NULL, sigma2 = NULL,
            var.equal = FALSE, alternative = c("two.sided", "less", "greater"),
             conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
             na.omit = FALSE, digits = 2, as.na = NULL, write = NULL,
             append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`data` a data frame. Note that data are specified either by using the argument `data` or by using the arguments `x` and `y`.

...	an expression indicating the variable names in data, e.g., ci.mean.diff(dat, x1, x2). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
x	a numeric vector of data values.
y	a numeric vector of data values.
formula	a formula of the form y ~ group for one outcome variable or cbind(y1, y2, y3) ~ group for more than one outcome variable where y is a numeric variable giving the data values and group a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
mu	a numeric value indicating the population mean under the null hypothesis. Note that the argument mu is only used when y = NULL.
sigma	a numeric vector indicating the population standard deviation(s) when computing confidence intervals for the difference in arithmetic means with known standard deviation(s). In case of independent samples, equal standard deviations are assumed when specifying one value for the argument sigma; when specifying two values for the argument sigma, unequal standard deviations are assumed. Note that either argument sigma or argument sigma2 is specified and it is only possible to specify one value (i.e., equal variance assumption) or two values (i.e., unequal variance assumption) for the argument sigma even though multiple variables are specified in x.
sigma2	a numeric vector indicating the population variance(s) when computing confidence intervals for the difference in arithmetic means with known variance(s). In case of independent samples, equal variances are assumed when specifying one value for the argument sigma2; when specifying two values for the argument sigma, unequal variances are assumed. Note that either argument sigma or argument sigma2 is specified and it is only possible to specify one value (i.e., equal variance assumption) or two values (i.e., unequal variance assumption) for the argument sigma even though multiple variables are specified in x.
var.equal	logical: if TRUE, the population variance in the independent samples are assumed to be equal.
paired	logical: if TRUE, confidence interval for the difference of arithmetic means in paired samples is computed.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	a numeric vector, character vector or factor as grouping variable. Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
split	a numeric vector, character vector or factor as split variable. Note that a split variable can only be used when computing confidence intervals with unknown population
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.

digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to x, but not to group or split.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	list with the input specified in x, group, and split
args	specification of function arguments
result	result table

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

test.z, test.t, ci.mean, ci.median, ci.prop, ci.var, ci.sd, descript

Examples

```
#————————————————————————————————————————————————————————————————————
# One-Sample Design

# Example 1a: Two-Sided 95% CI for 'mpg', population mean = 20
ci.mean.diff(mtcars, mpg, mu = 20)

# Example 1b: Alternative specification without using the '...' argument
ci.mean.diff(mtcars$mpg, mu = 20)

#————————————————————————————————————————————————————————————————————
# Two-Sample Design
```

```
# Example 2a: Two-Sided 95% CI for 'mpg' by 'vs'
ci.mean.diff(mpg ~ vs, data = mtcars)

# Example 2b: Alternative specification without using the '...' argument
ci.mean.diff(x = df.subset(mtcars, mpg, subset = vs == 0),
             y = df.subset(mtcars, mpg, subset = vs == 1))

# Example 2c: Equal variance assumption
ci.mean.diff(mpg ~ vs, data = mtcars, var.equal = TRUE)

# Example 2d: Known equal population standard deviations
ci.mean.diff(mpg ~ vs, data = mtcars, sigma = 4)

# Example 2e: Known unequal population standard deviations
ci.mean.diff(mpg ~ vs, data = mtcars, sigma = c(4, 5))

# Example 2f: Two-Sided 95% CI for 'mpg', 'cyl', and 'disp' by 'vs'
ci.mean.diff(cbind(mpg, cyl, disp) ~ vs, data = mtcars)

# Example 2g: Analysis by group separately
ci.mean.diff(cbind(mpg, cyl, disp) ~ vs, data = mtcars, group = "am")

# Example 2h: Analysis split by group
ci.mean.diff(cbind(mpg, cyl, disp) ~ vs, data = mtcars, split = "am")

#-----------------------------------------------------------------------
# Paired-Sample Design

# Example 3a: Two-Sided 95% CI for the mean difference in 'pre' and 'post'
ci.mean.diff(mtcars, drat, wt, paired = TRUE)

# Example 3a: Alternative specification without using the '...' argument
ci.mean.diff(x = mtcars$drat, y = mtcars$wt, paired = TRUE)

# Example 3b: Analysis by group separately
ci.mean.diff(mtcars, drat, wt, paired = TRUE, group = "vs")

# Example 3c: Analysis split by group
ci.mean.diff(mtcars, drat, wt, paired = TRUE, split = "vs")

# Example 3e: Known population standard deviation of difference scores
ci.mean.diff(mtcars, drat, wt, sigma = 1, paired = TRUE)
```

`ci.mean.w` *Within-Subject Confidence Interval for the Arithmetic Mean*

Description

This function computes difference-adjusted Cousineau-Morey within-subject confidence interval for the arithmetic mean.

Usage

```
ci.mean.w(data, ..., adjust = TRUE,
          alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
          na.omit = TRUE, digits = 2, as.na = NULL, write = NULL, append = TRUE,
          check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with numeric variables representing the levels of the within-subject factor, i.e., data are specified in wide-format (i.e., multivariate person level format).
`...`	an expression indicating the variable names in `data`, e.g., `ci.mean.w(dat, time1, time2, time3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`adjust`	logical: if `TRUE` (default), difference-adjustment for the Cousineau-Morey within-subject confidence intervals is applied.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`na.omit`	logical: if `TRUE` (default), incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`write`	a character string naming a text file with file extension `".txt"` (e.g., `"Output.txt"`) for writing the output into a text file.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

The Cousineau within-subject confidence interval (CI, Cousineau, 2005) is an alternative to the Loftus-Masson within-subject CI (Loftus & Masson, 1994) that does not assume sphericity or homogeneity of covariances. This approach removes individual differences by normalizing the raw scores using participant-mean centering and adding the grand mean back to every score:

$$Y_{ij}^{'} = Y_{ij} - \hat{\mu}_i + \hat{\mu}_{grand}$$

where $Y_{ij}^{'}$ is the score of the ith participant in condition j (for $i = 1$ to n), $\hat{\mu}_i$ is the mean of participant i across all J levels (for $j = 1$ to J), and $\hat{\mu}_{grand}$ is the grand mean.

Morey (2008) pointed out that Cousineau's (2005) approach produces intervals that are consistently too narrow due to inducing a positive covariance between normalized scores within a condition

introducing bias into the estimate of the sample variances. The degree of bias is proportional to the number of means and can be removed by rescaling the confidence interval by a factor of $\sqrt{J-1}/J$:

$$\hat{\mu}_j \pm t_{n-1,1-\alpha/2}\sqrt{\frac{J}{J-1}}\hat{\sigma}^{'}_{\hat{\mu}_j}$$

where $\hat{\sigma}^{'}_{\mu_j}$ is the standard error of the mean computed from the normalized scores of he jth factor level.

Baguley (2012) pointed out that the Cousineau-Morey interval is larger than that for a difference in means by a factor of $\sqrt{2}$ leading to a misinterpretation of these intervals that overlap of 95% confidence intervals around individual means is indicates that a 95% confidence interval for the difference in means would include zero. Hence, following adjustment to the Cousineau-Morey interval was proposed:

$$\hat{\mu}_j \pm \frac{\sqrt{2}}{2}(t_{n-1,1-\alpha/2}\sqrt{\frac{J}{J-1}}\hat{\sigma}^{'}_{\hat{\mu}_j})$$

The adjusted Cousineau-Morey interval is informative about the pattern of differences between means and is computed by default (i.e., `adjust = TRUE`).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`result`	result table

Author(s)

Takuya Yanagida

References

Baguley, T. (2012). Calculating and graphing within-subject confidence intervals for ANOVA. *Behavior Research Methods, 44*, 158-175. https://doi.org/10.3758/s13428-011-0123-7

Cousineau, D. (2005) Confidence intervals in within-subject designs: A simpler solution to Loftus and Masson's Method. *Tutorials in Quantitative Methods for Psychology, 1*, 42–45. https://doi.org/10.20982/tqmp.01.1.p042

Loftus, G. R., and Masson, M. E. J. (1994). Using confidence intervals in within-subject designs. *Psychonomic Bulletin and Review, 1*, 476–90. https://doi.org/10.3758/BF03210951

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to Cousineau. *Tutorials in Quantitative Methods for Psychology, 4*, 61–4. https://doi.org/10.20982/tqmp.01.1.p042

See Also

aov.w, test.z, test.t, ci.mean.diff,' ci.median, ci.prop, ci.var, ci.sd, descript

Examples

```
dat <- data.frame(time1 = c(3, 2, 1, 4, 5, 2, 3, 5, 6, 7),
                  time2 = c(4, 3, 6, 5, 8, 6, 7, 3, 4, 5),
                  time3 = c(1, 2, 2, 3, 6, 5, 1, 2, 4, 6))

# Example 1: Difference-adjusted Cousineau-Morey confidence intervals
ci.mean.w(dat)

# Example 2: Cousineau-Morey confidence intervals
ci.mean.w(dat, adjust = FALSE)

## Not run:
# Example 3: Write results into a text file
ci.mean.w(dat, write = "WS_Confidence_Interval.txt")
## End(Not run)
```

---

ci.prop *(Bootstrap) Confidence Intervals for Proportions*

Description

This function computes and plots confidence intervals for proportions, optionally by a grouping and/or split variable. The function also supports three types of bootstrap confidence intervals (e.g., bias-corrected (BC) percentile bootstrap or bias-corrected and accelerated (BCa) bootstrap confidence intervals) and plots the bootstrap samples with histograms and density curves.

Usage

```
ci.prop(data, ..., method = c("wald", "wilson"),
        boot = c("none", "perc", "bc", "bca"), nrep = 1000, seed = NULL,
        alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
        group = NULL, split = NULL, sort.var = FALSE, na.omit = FALSE,
        digits = 3, as.na = NULL, plot = c("none", "ci", "boot"),
        geom = c("all", "hist", "density", "point", "ci", "line"),
        filename = NULL, width = NA, height = NA, dpi = 600,
        write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

data	a numeric vector or data frame with numeric variables with 0 and 1 values.
...	an expression indicating the variable names in data, e.g., ci.prop(dat, x1, x2, x3). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.

method	a character string specifying the method for computing the confidence interval, must be one of "wald", or "wilson" (default).
boot	a character string specifying the type of bootstrap confidence intervals (CI), i.e., "none" (default) for not conducting bootstrapping, "perc", for the percentile bootstrap CI "bc" (default) for the bias-corrected (BC) percentile bootstrap CI (without acceleration), and "bca" for the bias-corrected and accelerated (BCa) bootstrap CI, see 'Details' in the ci.cor function.
nrep	a numeric value indicating the number of bootstrap replicates (default is 1000).
seed	a numeric value specifying seeds of the pseudo-random numbers used in the bootstrap algorithm when conducting bootstrapping.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	either a character string indicating the variable name of the grouping variable in data, or a vector representing the grouping variable.
split	either a character string indicating the variable name of the split variable in data, or a vector representing the split variable.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data, but not to group or split.
plot	a character string indicating the type of the plot to display, i.e., "none" (default) for not displaying any plots, "ci" for displaying confidence intervals for the proportion, "boot" for displaying bootstrap samples with histograms and density curves when the argument "boot" is other than "none".
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "hist" for drawing histograms when plotting bootstrap samples (plot = "boot"), "density" for drawing density curves when plotting bootstrap samples (plot = "boot"), "point" for drawing vertical lines representing the point estimate of the proportion when plotting bootstrap samples (plot = "boot"), "ci" for drawing vertical lines representing the bootstrap confidence intervals of the proportion when plotting bootstrap samples (plot = "boot"), and "line" for drawing a horizontal line when specifying plot = "ci" or a vertical line when specifying plot = "boot".
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = "ci" or plot = "boot".
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.

height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

The Wald confidence interval which is based on the normal approximation to the binomial distribution are computed by specifying method = "wald", while the Wilson (1927) confidence interval (aka Wilson score interval) is requested by specifying method = "wilson". By default, Wilson confidence interval is computed which have been shown to be reliable in small samples of n = 40 or less, and larger samples of n > 40 (Brown, Cai & DasGupta, 2001), while the Wald confidence intervals is inadequate in small samples and when p is near 0 or 1 (Agresti & Coull, 1998).

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	list with the input specified in ..., data, group, and split
args	specification of function arguments
boot	data frame with bootstrap replicates of the aproportion when bootstrapping was requested
plot	ggplot2 object for plotting the results and the data frame used for plotting
result	result table

Note

Bootstrap confidence intervals are computed using the R package boot by Angelo Canty and Brain Ripley (2024).

Author(s)

Takuya Yanagida

References

Agresti, A. & Coull, B.A. (1998). Approximate is better than "exact" for interval estimation of binomial proportions. *American Statistician, 52*, 119-126.

Brown, L. D., Cai, T. T., & DasGupta, A., (2001). Interval estimation for a binomial proportion. *Statistical Science, 16*, 101-133.

Canty, A., & Ripley, B. (2024). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-31.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association, 22*, 209-212.

See Also

ci.prop, ci.prop.diff, ci.median, ci.prop.diff, ci.cor, ci.var, ci.sd, descript

Examples

```
#═══════════════════════════════════════════════════════════════════════
# Confidence Interval (CI) for Proportions

# Example 1a: Two-Sided 95% CI
ci.prop(mtcars, vs, am)

# Example 1b: Alternative specification without using the '...' argument
ci.prop(mtcars[, c("vs", "am")])

# Example 1c: One-Sided 95% CI using Wald method
ci.prop(mtcars, vs, am, method = "wald", alternative = "less")

#═══════════════════════════════════════════════════════════════════════
# Grouping Variable

# Example 2a: Grouping variable
ci.prop(mtcars, vs, group = "am")

# Example 2b: Alternative specification without using the '...' argument
ci.prop(mtcars$vs, group = mtcars$am)

# Example 2c: Sort result table by variables
ci.prop(mtcars, vs, am, group = "gear", sort.var = TRUE)

#═══════════════════════════════════════════════════════════════════════
# Split Variable

# Example 3a: Split variable
ci.prop(mtcars, vs, split = "am")

# Example 3b: Alternative specification without using the '...' argument
ci.prop(mtcars$vs, split = mtcars$am)
```

```
#═══════════════════════════════════════════════════════════════════════════
# Grouping and Split Variable

# Example 4a: Grouping and split variable
ci.prop(mtcars, vs, group = "am", split = "cyl")

# Example 4b: Alternative specification without using the '...' argument
ci.prop(mtcars$vs, group = mtcars$am, split = mtcars$cyl)

# Example 4c: Sort result tables by variables
ci.prop(mtcars, vs, am, group = "gear", split = "cyl", sort.var = TRUE)

## Not run:
#═══════════════════════════════════════════════════════════════════════════
# Bootstrap Confidence Interval (CI)

# Example 5a: Bias-corrected (BC) percentile bootstrap CI
ci.prop(mtcars, vs, am, boot = "bc")

# Example 5b: Bias-corrected and accelerated (BCa) bootstrap CI,
# 5000 bootstrap replications, set seed of the pseudo-random number generator
ci.prop(mtcars, vs, am, boot = "bca", nrep = 5000, seed = 42)

#═══════════════════════════════════════════════════════════════════════════
# Plot Confidence Intervals

# Example 6a: Two-Sided 95
ci.prop(mtcars, vs, am, plot = "ci")

# Example 6b: Grouping variable
ci.prop(mtcars, vs, am, group = "am", plot = "ci")

# Example 6c: Split variable
ci.prop(mtcars, vs, am, split = "am", plot = "ci")

# Example 6d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.prop(mtcars, vs, am, plot = "ci")
plot(object, ybreaks = seq(0, 1, by = 0.1), title = "Confidence Intervals")

#═══════════════════════════════════════════════════════════════════════════
# Plot Bootstrap Samples

# Example 7a: Two-Sided 95
ci.prop(mtcars, vs, am, boot = "bc", plot = "boot")

# Example 7b: Grouping variable
ci.prop(mtcars, vs, am, group = "am", boot = "bc", plot = "boot")

# Example 7c: Split variable
ci.prop(mtcars, vs, am, split = "am", boot = "bc", plot = "boot")

# Example 7d: Plot results using the plot() function, use additional arguments
```

```
# see Details in the help page of the function plot.misty.object
object <- ci.prop(mtcars, vs, am, boot = "bc", plot = "boot")
plot(object, fill = "gray30", title = "Bootstrap Samples")

#------------------------------------------------------------------------
# Write Results and Save Plot

# Example 8a: Write results into a text file
ci.prop(mtcars, vs, am, write = "CI_Prop.txt")

# Example 8b: Write results into an Excel file
ci.prop(mtcars, vs, am, write = "CI_Prop.xlsx")

# Example 8c: Save plot as PNG file
ci.prop(mtcars, vs, am, plot = "ci", filename = "CI_Prop.png",
        width = 9, height = 6)

## End(Not run)
```

`ci.prop.diff` *Confidence Interval for the Difference in Proportions*

Description

This function computes confidence intervals for the difference in proportions in a two-sample and paired-sample design for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.prop.diff(...)

## Default S3 method:
ci.prop.diff(data = NULL, ..., x = NULL, y = NULL,
             method = c("wald", "newcombe"), paired = FALSE,
            alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
           group = NULL, split = NULL, sort.var = FALSE, digits = 2, as.na = NULL,
             write = NULL, append = TRUE, check = TRUE, output = TRUE)

## S3 method for class 'formula'
ci.prop.diff(formula, data, method = c("wald", "newcombe"),
            alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
             group = NULL, split = NULL, sort.var = FALSE, na.omit = FALSE,
             digits = 2, as.na = NULL, write = NULL, append = TRUE,
             check = TRUE, output = TRUE, ...)
```

Arguments

`data` a data frame. Note that data are specified either by using the argument `data` or by using the arguments `x` and `y`.

...	an expression indicating the variable names in data, e.g., ci.mean.diff(dat, x1, x2). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
x	a numeric vector of data values.
y	a numeric vector of data values.
formula	a formula of the form y ~ group for one outcome variable or cbind(y1, y2, y3) ~ group for more than one outcome variable where y is a numeric variable giving the data values and group a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
method	a character string specifying the method for computing the confidence interval, must be one of "wald", or "newcombe" (default).
paired	logical: if TRUE, confidence interval for the difference of proportions in paired samples is computed.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	a numeric vector, character vector or factor as grouping variable. Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
split	a numeric vector, character vector or factor as split variable. Note that a split variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to x, but not to group or split.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

The Wald confidence interval which is based on the normal approximation to the binomial distribution are computed by specifying method = "wald", while the Newcombe Hybrid Score interval (Newcombe, 1998a; Newcombe, 1998b) is requested by specifying method = "newcombe". By default, Newcombe Hybrid Score interval is computed which have been shown to be reliable in small samples (less than n = 30 in each sample) as well as moderate to larger samples(n > 30 in each sample) and with proportions close to 0 or 1, while the Wald confidence intervals does not perform well unless the sample size is large (Fagerland, Lydersen & Laake, 2011).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the input specified in `x`, `group`, and `split`
`args`	specification of function arguments
`result`	result table

Author(s)

Takuya Yanagida

References

Fagerland, M. W., Lydersen S., & Laake, P. (2011) Recommended confidence intervals for two independent binomial proportions. *Statistical Methods in Medical Research, 24*, 224-254.

Newcombe, R. G. (1998a). Interval estimation for the difference between independent proportions: Comparison of eleven methods. *Statistics in Medicine, 17*, 873-890.

Newcombe, R. G. (1998b). Improved confidence intervals for the difference between binomial proportions based on paired data. *Statistics in Medicine, 17*, 2635-2650.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

`ci.prop`, `ci.mean`, `ci.mean.diff`, `ci.median`, `ci.var`, `ci.sd`, `descript`

Examples

```
#————————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 1a: Two-Sided 95% CI for 'vs' by 'am', Newcombes Hybrid Score interval
ci.prop.diff(vs ~ am, data = mtcars)

# Example 2b: Alternative specification without using the '...' argument
ci.mean.diff(x = df.subset(mtcars, vs, subset = am == 0),
             y = df.subset(mtcars, vs, subset = am == 1))

# Example 1c: Wald CI
ci.prop.diff(vs ~ am, data = mtcars, method = "wald")

#————————————————————————————————————————————————————————————————————
# Paired-Sample Design

# Example 2a: Two-Sided 95% CI for the difference in proportions 'pre' and 'post'
# Newcombes Hybrid Score interval
```

```
ci.prop.diff(mtcars, vs, am, paired = TRUE)

# Example 2b:  Alternative specification without using the '...' argument
ci.prop.diff(x = mtcars$vs, y = mtcars$am, paired = TRUE)

# Example 2b: Wald CI
ci.prop.diff(mtcars, vs, am, method = "wald", paired = TRUE)
```

`ci.var` *(Bootstrap) Confidence Intervals for Variances and Standard Deviations*

Description

The function `ci.var` computes and plots confidence intervals for variances, and the function `ci.sd` computes confidence intervals for the standard deviations, optionally by a grouping and/or split variable. These functions also supports three types of bootstrap confidence intervals (e.g., bias-corrected (BC) percentile bootstrap or bias-corrected and accelerated (BCa) bootstrap confidence intervals) and plots the bootstrap samples with histograms and density curves.

Usage

```
ci.var(data, ..., method = c("chisq", "bonett"),
       boot = c("none", "perc", "bc", "bca"), nrep = 1000, seed = NULL,
       alternative = c("two.sided", "less", "greater"),
       conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
      na.omit = FALSE, digits = 2, as.na = NULL, plot = c("none", "ci", "boot"),
       geom = c("all", "hist", "density", "point", "ci", "line"),
       filename = NULL, width = NA, height = NA, dpi = 600, write = NULL,
       append = TRUE, check = TRUE, output = TRUE)

ci.sd(data, ..., method = c("chisq", "bonett"),
      boot = c("none", "perc", "bc", "bca"), nrep = 1000, seed = NULL,
      alternative = c("two.sided", "less", "greater"),
      conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
      na.omit = FALSE, digits = 2, as.na = NULL, plot = c("none", "ci", "boot"),
      geom = c("all", "hist", "density", "point", "ci", "line"),
      filename = NULL, width = NA, height = NA, dpi = 600, write = NULL,
      append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a numeric vector or data frame with numeric variables, i.e., factors and character variables are excluded from `data` before conducting the analysis.
`...`	an expression indicating the variable names in `data`, e.g., `ci.var(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

method	a character string specifying the method for computing the confidence interval, must be one of "chisq", or "bonett" (default).
boot	a character string specifying the type of bootstrap confidence intervals (CI), i.e., "none" (default) for not conducting bootstrapping, "perc", for the percentile bootstrap CI "bc" (default) for the bias-corrected (BC) percentile bootstrap CI (without acceleration), and "bca" for the bias-corrected and accelerated (BCa) bootstrap CI, see 'Details' in the ci.cor function.
nrep	a numeric value indicating the number of bootstrap replicates (default is 1000).
seed	a numeric value specifying seeds of the pseudo-random numbers used in the bootstrap algorithm when conducting bootstrapping.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	either a character string indicating the variable name of the grouping variable in data, or a vector representing the grouping variable.
split	either a character string indicating the variable name of the split variable in 'data', or a vector representing the split variable.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data, but not to group or split.
plot	a character string indicating the type of the plot to display, i.e., "none" (default) for not displaying any plots, "ci" for displaying confidence intervals for variances or standard deviations, "boot" for displaying bootstrap samples with histograms and density curves when the argument "boot" is other than "none".
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "hist" for drawing histograms when plotting bootstrap samples (plot = "boot"), "density" for drawing density curves when plotting bootstrap samples (plot = "boot"), "point" for drawing vertical lines representing the point estimate of the variance or standard deviation when plotting bootstrap samples (plot = "boot"), "ci" for drawing vertical lines representing the bootstrap confidence intervals of the variance or standard deviation when plotting bootstrap samples (plot = "boot"), and "line" for drawing a horizontal line when specifying plot = "ci" or a vertical line when specifying plot = "boot".
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = "ci" or plot = "boot".
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.

height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

The confidence interval based on the chi-square distribution is computed by specifying method = "chisq", while the Bonett (2006) confidence interval is requested by specifying method = "bonett". By default, the Bonett confidence interval interval is computed which performs well under moderate departure from normality, while the confidence interval based on the chi-square distribution is highly sensitive to minor violations of the normality assumption and its performance does not improve with increasing sample size. Note that at least four valid observations are needed to compute the Bonett confidence interval.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	list with the input specified in ..., data, group, and split
args	specification of function arguments
boot	data frame with bootstrap replicates of the variance or standard deviation when bootstrapping was requested
plot	ggplot2 object for plotting the results and the data frame used for plotting
result	result table

Note

Bootstrap confidence intervals are computed using the R package boot by Angelo Canty and Brain Ripley (2024).

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Canty, A., & Ripley, B. (2024). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-31.

Bonett, D. G. (2006). Approximate confidence interval for standard deviation of nonnormal distributions. *Computational Statistics and Data Analysis, 50*, 775-782. https://doi.org/10.1016/j.csda.2004.10.003

See Also

ci.mean, ci.mean.diff, ci.median, ci.prop, ci.prop.diff, ci.cor, descript

Examples

```
#-----------------------------------------------------------------------
# Confidence Interval (CI) for the Variance

# Example 1a: Two-Sided 95% CI
ci.var(mtcars)

# Example 1b: One-Sided 99% CI based on the chi-square distributio
ci.var(mtcars, alternative = "less", method = "chisq")

#-----------------------------------------------------------------------
# Confidence Interval (CI) for the Standard Deviation

# Example 2a: Two-Sided 95% CI
ci.sd(mtcars)

# Example 2b: One-Sided 99% CI based on the chi-square distribution
ci.sd(mtcars, alternative = "less", method = "chisq")

#-----------------------------------------------------------------------
# Grouping Variable

# Example 3a: Grouping variable
ci.var(mtcars, mpg, cyl, disp, group = "vs")

# Example 3b:  Alternative specification without using the '...' argument
ci.var(mtcars[, c("mpg", "cyl", "disp")], group = mtcars$vs)

# Example 3c: Sort result tables by variables
ci.var(mtcars, mpg, cyl, disp, group = "vs", sort.var = TRUE)

#-----------------------------------------------------------------------
# Split Variable

# Example 4a: Split variable
ci.var(mtcars, mpg, cyl, disp, split = "am")

# Example 4b:  Alternative specification without using the '...' argument
ci.var(mtcars[, c("mpg", "cyl", "disp")], split = mtcars$am)
```

```
#————————————————————————————————————————————————————————————————————————
# Grouping and Split Variable

# Example 5a: Grouping and split variable
ci.var(mtcars, mpg, cyl, disp, group = "vs", split = "am")

# Example 5b:  Alternative specification without using the '...' argument
ci.var(mtcars[, c("mpg", "cyl", "disp")], group = mtcars$vs, split = mtcars$am)

# Example 5c: Sort result tables by variables
ci.var(mtcars, mpg, cyl, disp, group = "vs", split = "am", sort.var = TRUE)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Bootstrap Confidence Interval (CI)

# Example 6a: Bias-corrected (BC) percentile bootstrap CI
ci.var(mtcars, boot = "bc")

# Example 6b: Bias-corrected and accelerated (BCa) bootstrap CI,
# 5000 bootstrap replications, set seed of the pseudo-random number generator
ci.var(mtcars, boot = "bca", nrep = 5000, seed = 42)

#————————————————————————————————————————————————————————————————————————
# Plot Confidence Intervals

# Example 7a: Two-Sided 95
ci.var(mtcars, plot = "ci")

# Example 7b: Grouping variable
ci.var(mtcars, disp, hp, group = "vs", plot = "ci")

# Example 7c: Split variable
ci.var(mtcars, disp, hp, split = "am", plot = "ci")

# Example 7d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.var(mtcars, disp, hp, plot = "ci")
plot(object, ybreaks = seq(0, 25000, by = 2500), title = "Confidence Intervals")

#————————————————————————————————————————————————————————————————————————
# Plot Bootstrap Samples

# Example 8a: Two-Sided 95
ci.var(mtcars, disp, hp, boot = "bc", plot = "boot")

# Example 8b: Grouping variable
ci.var(mtcars, disp, hp, group = "vs", boot = "bc", plot = "boot")

# Example 8c: Split variable
ci.var(mtcars, disp, hp, split = "am", boot = "bc", plot = "boot")
```

```
# Example 8d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- ci.var(mtcars, disp, hp, boot = "bc", plot = "boot")
plot(object, fill = "gray30", title = "Bootstrap Samples")

#-----------------------------------------------------------------------
# Write Results and Save Plot

# Example 9a: Write results into a text file
ci.var(mtcars, disp, write = "CI_Var.txt")

# Example 9b: Write results into an Excel file
ci.var(mtcars, disp, write = "CI_Var.xlsx")

# Example 9c: Save plot as PNG file
ci.var(mtcars, disp, plot = "ci", filename = "CI_Var.png",
        width = 9, height = 6)

## End(Not run)
```

`clear` *Clear Console in RStudio*

Description

This function clears the console equivalent to `Ctrl + L` in RStudio on Windows, Mac, UNIX, or Linux operating system.

Usage

```
clear()
```

Author(s)

Takuya Yanagida

See Also

`restart`, `setsource`

Examples

```
## Not run:

# Clear console
clear()

## End(Not run)
```

`cluster.rwg` *Lindell, Brandt and Whitney (1999) r*wg(j) Within-Group Agreement Index for Multi-Item Scales*

Description

This function computes r*wg(j) within-group agreement index for multi-item scales as described in Lindell, Brandt and Whitney (1999).

Usage

```
cluster.rwg(data, ...,  cluster, A = NULL, ranvar = NULL, z = TRUE,
            expand = TRUE, na.omit = FALSE, append = TRUE, name = "rwg",
            as.na = NULL, check = TRUE)
```

Arguments

`data`	a numeric vector or data frame.
`...`	an expression indicating the variable names in `data`, e.g., `cluster.rwg(dat, x1, x2, x3)`. Note that the operators `.`, `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`cluster`	either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).
`A`	a numeric value indicating the number of discrete response options of the items from which the random variance is computed based on $(A^2 - 1)/12$. Note that either the argument `j` or the argument`ranvar` is specified.
`ranvar`	a numeric value indicating the random variance to which the mean of the item variance is divided. Note that either the argument `j` or the argument`ranvar` is specified.
`z`	logical: if `TRUE` (default), Fisher z-transformation based on the formula $z = 0.5 * log((1 + r)/(1 - r))$ is applied to the vector of r*wg(j) estimates.
`expand`	logical: if `TRUE` (default), vector of r*wg(j) estimates is expanded to match the input vector `data`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`append`	logical: if `TRUE` (default), a variable with the r*wg(j) within-group agreement index are appended to the data frame specified in the argument `data`.
`name`	a character string indicating the name of the variable appended to the data frame specified in the argument `data` when `append = TRUE`. By default, the variable is named `rwg`.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data`, but not to `cluster`.
`check`	logical: if `TRUE` (default), argument specification is checked.

Details

The r*wg(j) index is calculated by dividing the mean of the item variance by the expected random variance (i.e., null distribution). The default null distribution in most research is the rectangular or uniform distribution calculated with $\sigma_e^2 u = (A^2 - 1)/12$, where A is the number of discrete response options of the items. However, what constitutes a reasonable standard for random variance is highly debated. Note that the r*wg(j) allows that the mean of the item variances to be larger than the expected random variances, i.e., r*wg(j) values can be negative.

Note that the `rwg.j.lindell()` function in the **multilevel** package uses listwise deletion by default, while the `cluster.rwg()` function uses all available information to compute the r*wg(j) agreement index by default. In order to obtain equivalent results in the presence of missing values, listwise deletion (`na.omit = TRUE`) needs to be applied.

Value

Returns a numeric vector containing r*wg(j) agreement index for multi-item scales with the same length as `group` if `expand = TRUE` or a data frame with following entries if `expand = FALSE`:

`cluster`	cluster identifier
`n`	cluster size
`rwg.lindell`	r*wg(j) estimate for each group
`z.rwg.lindell`	Fisher z-transformed r*wg(j) estimate for each cluster

Author(s)

Takuya Yanagida

References

Lindell, M. K., Brandt, C. J., & Whitney, D. J. (1999). A revised index of interrater agreement for multi-item ratings of a single target. *Applied Psychological Measurement*, *23*, 127-135. https://doi.org/10.1177/01466219922031257

O'Neill, T. A. (2017). An overview of interrater agreement on Likert scales for researchers and practitioners. *Frontiers in Psychology*, *8*, Article 777. https://doi.org/10.3389/fpsyg.2017.00777

See Also

`cluster.scores`

Examples

```
dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                  cluster = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                  x1 = c(2, 3, 2, 1, 1, 2, 4, 3, 5),
                  x2 = c(3, 2, NA, 1, 2, 1, 3, 2, 5),
                  x3 = c(3, 1, 1, 2, 3, 3, 5, 5, 4))

# Example 1: Compute Fisher z-transformed r*wg(j) for a multi-item scale with A = 5 response options
cluster.rwg(dat, x1, x2, x3, cluster = "cluster", A = 5)
```

```
# Alternative specification without using the '...' argument
cluster.rwg(dat[, c("x1", "x2", "x3")], cluster = dat$cluster, A = 5)

# Example 2: Compute Fisher z-transformed r*wg(j) for a multi-item scale with a random variance of 2
cluster.rwg(dat, x1, x2, x3, cluster = "cluster", ranvar = 2)

# Example 3: Compute r*wg(j) for a multi-item scale with A = 5 response options
cluster.rwg(dat, x1, x2, x3, cluster = "cluster", A = 5, z = FALSE)

# Example 4: Do not expand Fisher z-transformed r*wg(j)
cluster.rwg(dat, x1, x2, x3, cluster = "cluster", A = 5, expand = FALSE)
```

`cluster.scores` *Cluster Scores*

Description

This function computes group means by default.

Usage

```
cluster.scores(data, ..., cluster,
               fun = c("mean", "sum", "median", "var", "sd", "min", "max"),
               expand = TRUE, append = TRUE, name = ".a", as.na = NULL,
               check = TRUE)
```

Arguments

`data`	a numeric vector for centering a predictor variable, or a data frame for centering more than one predictor variable.
`...`	an expression indicating the variable names in `data` e.g., `cluster.scores(dat, x1, x2, cluster = "cluster")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`cluster`	a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).
`fun`	character string indicating the function used to compute group scores, default: `"mean"`.
`expand`	logical: if `TRUE` (default), vector of cluster scores is expanded to match the input vector `data`.
`append`	logical: if `TRUE` (default), cluster scores are appended to the data frame specified in the argument `data`.
`name`	a character vector indicating the names of the computed variables. By default, variables are named with the ending `".a"` resulting in e.g. `"x1.a"` and `"x2.a"`. Variable names can also be specified using a character vector matching the number of variables specified in `data` (e.g., `name = c("cluster.x1", "cluster.x2")`).

as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to the argument data, but not to cluster.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a numeric vector or data frame containing cluster scores with the same length or same number of rows as data if expand = TRUE or with the length or number of rows as length(unique(cluster)) if expand = FALSE.

Author(s)

Takuya Yanagida

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

item.scores, multilevel.descript, multilevel.icc

Examples

```
# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Example 1: Compute cluster means for 'y1' and expand to match the input 'y1'
cluster.scores(Demo.twolevel, y1, cluster = "cluster", append = FALSE)

# Alternative specification without using the '...' argument
cluster.scores(Demo.twolevel$y1, cluster = Demo.twolevel$cluster)

# Example 2: Compute standard deviation for each cluster
# and expand to match the input x
cluster.scores(Demo.twolevel, cluster = "cluster", fun = "sd")

# Example 3: Compute cluster means without expanding the vector
cluster.scores(Demo.twolevel, cluster = "cluster", expand = FALSE)

# Example 4: Compute cluster means for 'y1' and 'y2' and append to 'Demo.twolevel'
cluster.scores(Demo.twolevel, y1, y2, cluster = "cluster")

# Alternative specification without using the '...' argument
cbind(Demo.twolevel,
      cluster.scores(Demo.twolevel[, c("y1", "y2")], cluster = Demo.twolevel$cluster))
```

coding *Coding Categorical Variables*

Description

This function creates $k-1$ variables for a categorical variable with k distinct levels. The coding system available in this function are dummy coding, simple coding, unweighted effect coding, weighted effect coding, repeated coding, forward Helmert coding, reverse Helmert coding, and orthogonal polynomial coding.

Usage

```
coding(data, ...,
       type = c("dummy", "simple", "effect", "weffect", "repeat",
                "fhelm", "rhelm", "poly"), base = NULL,
     name = c("dum.", "sim.", "eff.", "weff.", "rep.", "fhelm.", "rhelm.", "poly."),
       append = TRUE, as.na = NULL, check = TRUE)
```

Arguments

`data`	a numeric vector with integer values, character vector or factor.
`...`	an expression indicating the variable name in `data`, e.g., `coding(dat, x)`. Note that the function can only deal with one categorical variable.
`type`	a character string indicating the type of coding, i.e., `dummy` (default) for dummy coding, `simple` for simple coding, `effect` for unweighted effect coding, `weffect` for weighted effect coding, `repeat` for repeated coding, `fhelm` for forward Helmert coding, `rhelm` for reverse Helmert coding, and `poly` for orthogonal polynomial coding (see 'Details').
`base`	a numeric value or character string indicating the baseline group for dummy and simple coding and the omitted group in effect coding. By default, the first group or factor level is selected as baseline or omitted group.
`name`	a character vector indicating the names of the coded variables. By default, variables are named `"dum."`, `"sim."`, `"eff."`, `"weff."`, `"rep."`, `"fhelm."`, `"rhelm."`,or `"poly."` depending on the `type` of coding with the category used in the comparison (e.g., `"dum.2"` and `"dum.3"`). Variable names can be specified using a character string (e.g., `name = "dummy_"` leads to `dummy_2` and `dummy_3`) or a character vector matching the number of coded variables (e.g. `name = c("x1_2", "x1_3")`) which is the number of unique categories minus one.
`append`	logical: if `TRUE` (default), coded variables are appended to the data frame specified in the argument `data`.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE` (default), argument specification is checked.

Details

Dummy Coding Dummy or treatment coding compares the mean of each level of the categorical variable to the mean of a baseline group. By default, the first group or factor level is selected as baseline group. The intercept in the regression model represents the mean of the baseline group. For example, dummy coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs A`, `C vs A`, and `D vs A` with `A` being the baseline group.

Simple Coding Simple coding compares each level of the categorical variable to the mean of a baseline level. By default, the first group or factor level is selected as baseline group. The intercept in the regression model represents the unweighted grand mean, i.e., mean of group means. For example, simple coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs A`, `C vs A`, and `D vs A` with `A` being the baseline group.

Unweighted Effect Coding Unweighted effect or sum coding compares the mean of a given level to the unweighted grand mean, i.e., mean of group means. By default, the first group or factor level is selected as omitted group. For example, effect coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs (A, B, C, D)`, `C vs (A, B, C, D)`, and `D vs (A, B, C, D)` with `A` being the omitted group.

Weighted Effect Coding Weighted effect or sum coding compares the mean of a given level to the weighed grand mean, i.e., sample mean. By default, the first group or factor level is selected as omitted group. For example, effect coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs (A, B, C, D)`, `C vs (A, B, C, D)`, and `D vs (A, B, C, D)` with `A` being the omitted group.

Repeated Coding Repeated or difference coding compares the mean of each level of the categorical variable to the mean of the previous adjacent level. For example, repeated coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs A`, `C vs B`, and `D vs C`.

Foward Helmert Coding Forward Helmert coding compares the mean of each level of the categorical variable to the unweighted mean of all subsequent level(s) of the categorical variable. For example, forward Helmert coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `(B, C, D) vs A`, `(C, D) vs B`, and `D vs C`.

Reverse Helmert Coding Reverse Helmert coding compares the mean of each level of the categorical variable to the unweighted mean of all prior level(s) of the categorical variable. For example, reverse Helmert coding based on a categorical variable with four groups `A`, `B`, `C`, `D` makes following comparisons: `B vs A`, `C vs (A, B)`, and `D vs (A, B, C)`.

Orthogonal Polynomial Coding Orthogonal polynomial coding is a form of trend analysis based on polynomials of order $k - 1$, where k is the number of levels of the categorical variable. This coding scheme assumes an ordered-categorical variable with equally spaced levels. For example, orthogonal polynomial coding based on a categorical variable with four groups `A`, `B`, `C`, `D` investigates a linear, quadratic, and cubic trends in the categorical variable.

Value

Returns a data frame with $k - 1$ coded variables or a data frame with the same length or same number of rows as ... containing the coded variables.

Note

This function uses the `contr.treatment` function from the **stats** package for dummy coding and simple coding, a modified copy of the `contr.sum` function from the **stats** package for effect coding, a modified copy of the `contr.wec` function from the **wec** package for weighted effect coding, a modified copy of the `contr.sdif` function from the **MASS** package for repeated coding, a modified copy of the `code_helmert_forward` function from the **codingMatrices** for forward Helmert coding, a modified copy of the `contr_code_helmert` function from the **faux** package for reverse Helmert coding, and the `contr.poly` function from the **stats** package for orthogonal polynomial coding.

Author(s)

Takuya Yanagida

See Also

`rec`, `item.reverse`

Examples

```
# Example 1: Dummy coding for 'gear', baseline group = 3
coding(mtcars, gear)

# Alternative specification without using the '...' argument
coding(mtcars$gear)

# Example 2: Dummy coding for 'gear', baseline group = 4
coding(mtcars, gear, base = 4)

# Example 3: Effect coding for 'gear', omitted group = 3
coding(mtcars, gear, type = "effect")

# Example 3: Effect coding for 'gear', omitted group = 4
coding(mtcars, gear, type = "effect", base = 4)

# Example 4a: Dummy-coded variable names with prefix "gear3."
coding(mtcars, gear, name = "gear3.")

# Example 4b: Dummy-coded variables named "gear_4vs3" and "gear_5vs3"
coding(mtcars, gear, name = c("gear_4vs3", "gear_5vs3"))
```

`coeff.robust` *Heteroscedasticity-Consistent and Cluster-Robust Standard Errors*

Description

This function computes (1) heteroscedasticity-consistent or cluster-robust standard errors standard errors and significance values for (generalized) linear models estimated by using the `lm()` or the `glm()` function and (2) cluster-robust standard errors for multilevel and linear mixed-effects models estimated by using the `lmer()` function from the **lme4** package or by using the `lme()` function from the **nlme** package that are robust to the violation of the homoscedasticity assumption. For linear models the heteroscedasticity-robust F-test is computed as well. By default, the function uses the HC4 estimator for (generalized) linear models and the heteroscedastic-robust CR2 estimator for multilevel and linear mixed-effects models. Note that cluster-robust standard errors are available only for two-level models.

Usage

```
coeff.robust(model, cluster = NULL,
             type = c("HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5",
                      "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3"),
             digits = 2, p.digits = 3, write = NULL, append = TRUE,
             check = TRUE, output = TRUE)
```

Arguments

`model` a fitted model of class `lm`, `glm`, `lmerMod`, `lmerModLmerTest`, or `lme`.

`cluster` a vector representing the nested grouping structure (i.e., group or cluster variable). This argument is used only when requesting cluster-robust standard errors for (generalized) linear models estimated by using the `lm()` or `glm()` function. Note that the length of the vector needs to match the number of rows of the data frame used to estimate the (generalized) linear model. In the presence of missing data, the length of the vector needs to match the number of rows of the data frame after listwise deletion of missing data.

`type` a character string specifying the estimation type for (generalized) linear models estimated by using the `lm()` or `glm()` function, where `"H0"` gives White's estimator and `"H1"` to `"H5"` are refinement of this estimator. See help page of the `vcovHC()` function in the R package `sandwich` for more details. Alternatively, a character string specifying the estimation type for multilevel and linear mixed-effects model estimated by using the `lmer()` function from the **lme4** package, where `"CR0"` is the original form of the sandwich estimator (Liang & Zeger, 1986), `"CR1"` multiplies CR0 by $m/(m-1)$, where m is the number of clusters, `"CR1p"` multiplies CR0 by $m/(m-p)$, where p is the number of covariates, `"CR1S"` multiplies CR0 by $(m(N-1))/[(m-1)(N-p)]$, where N is the total number of observations, `"CR2"` (default) is the "bias-reduced linearization" adjustment proposed by Bell and McCaffrey (2002) and further developed in Pustejovsky and Tipton (2017), and `"CR3"` approximates the leave-one-cluster-out jackknife variance estimator (Bell & McCaffrey, 2002). See help page of the `vcovCR()` function in the R package `clubSandwich` for more details.

`digits` an integer value indicating the number of decimal places to be used for displaying results. Note that information criteria and chi-square test statistic are printed with `digits` minus 1 decimal places.

`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Details

Heteroscedasticity-Consistent Standard Errors The family of heteroscedasticity-consistent (HC) standard errors estimator for the model parameters of a regression model is based on an HC covariance matrix of the parameter estimates and does not require the assumption of homoscedasticity. HC estimators approach the correct value with increasing sample size, even in the presence of heteroscedasticity. On the other hand, the OLS standard error estimator is biased and does not converge to the proper value when the assumption of homoscedasticity is violated (Darlington & Hayes, 2017). White (1980) introduced the idea of HC covariance matrix to econometricians and derived the asymptotically justified form of the HC covariance matrix known as HC0 (Long & Ervin, 2000). Simulation studies have shown that the HC0 estimator tends to underestimate the true variance in small to moderately large samples ($N \leq 250$) and in the presence of leverage observations, which leads to an inflated type I error risk (e.g., Cribari-Neto & Lima, 2014). The alternative estimators HC1 to HC5 are asymptotically equivalent to HC0 but include finite-sample corrections, which results in superior small sample properties compared to the HC0 estimator. Long and Ervin (2000) recommended routinely using the HC3 estimator regardless of a heteroscedasticity test. However, the HC3 estimator can be unreliable when the data contains leverage observations. The HC4 estimator, on the other hand, performs well with small samples, in the presence of high leverage observations, and when errors are not normally distributed (Cribari-Neto, 2004). In summary, it appears that the HC4 estimator performs the best in terms of controlling the type I and type II error risk (Rosopa, 2013). As opposed to the findings of Cribari-Neto et al. (2007), the HC5 estimator did not show any substantial advantages over HC4. Both HC5 and HC4 performed similarly across all the simulation conditions considered in the study (Ng & Wilcox, 2009). Note that the F-test of significance on the multiple correlation coefficient R also assumes homoscedasticity of the errors. Violations of this assumption can result in a hypothesis test that is either liberal or conservative, depending on the form and severity of the heteroscedasticity. Hayes (2007) argued that using a HC estimator instead of assuming homoscedasticity provides researchers with more confidence in the validity and statistical power of inferential tests in regression analysis. Hence, the HC3 or HC4 estimator should be used routinely when estimating regression models. If a HC estimator is not used as the default method of standard error estimation, researchers are advised to at least double-check the results by using an HC estimator to ensure that conclusions are not compromised by heteroscedasticity. However, the presence of heteroscedasticity suggests that the data is not adequately explained by the statistical model of estimated conditional means. Unless heteroscedasticity is believed to be solely caused by measurement error associated with the predictor variable(s), it should serve as warning to the researcher regarding the adequacy of the estimated model.

Cluster-Robust Standard Errors The family of cluster-robust (CR) standard errors estimator for the model parameters of a multilevel and linear mixed-effects model are based on the heteroscedasticity-consistent (HC) standard errors estimators that have been generalized to clustered data (Zhang & Lai, 2024). The standard errors of the CR0 estimator (Liang and Zeger, 1986) rely on large samples, i.e., the CR0 estimator may result in underestimated standard errors with small number of clusters (Cameron & Miller, 2015; Imbens & Kolesar, 2016). However, there is no consensus about the minimum number of clusters, e.g., at least 100 clusters (Maas & Hox, 2004, p. 439), around 40 (Angrist & Pischke, 2008) or 30 clusters (Huang, 2016). The CR2 estimator, also referred to as Bell and McCaffrey (2002) bias-reduced linearization method, has been shown to be effective when used with a small number of clusters (Hugang & Li, 2022). For example, the CR2 estimator performed well in all conditions of a simulation study involving 20, 50, or 100 clusters regardless if homoskedasticity was violated or not. (Huang, et al, 2023). The CR3 estimator tends to over-correct the bias of the CR0 estiamator, while the CR1 estimator tends to under-correct the bias (Pustejovsky & Tipton, 2018). Note that the cluster-robust SE are only robust to violation of the homoscedasticity assumption, while departure from normality or the presence of outliers can influence its performance (MacKinnon, 2012). Statistical significance testing of the regression coefficients is based on the Satterthwaite approximated degrees of freedom (Bell & McCaffrey (2002).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`args`	specification of function arguments
`result`	list with results, i.e., `coef` for the unstandardized regression coefficients with heteroscedasticity-consistent or cluster-robust standard errors, `F.test` for the heteroscedasticity-robust F-Test, and `sandwich` for the sandwich covariance matrix

Note

The computation of heteroscedasticity-consistent standard errors is based on the `vcovHC` function from the **sandwich** package (Zeileis, Köll, & Graham, 2020) and the functions `coeftest` and `waldtest` from the `lmtest` package (Zeileis & Hothorn, 2002), while the computation of cluster-robust standard errors uses the `vcovCR` and the `coef_test` function in the **clubSandwich** package.

Author(s)

Takuya Yanagida

References

Angrist, J. D., & Pischke, J.-S. (2008). *Mostly harmless econometrics: An empiricist's companion*. Princeton university press.

Bell, R. M., & McCaffrey, D. F. (2002). Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology, 28*(2), 169-181

Cameron, A. C., & Miller, D. L. (2015). A practitioner's guide to cluster-robust inference. *Journal of Human Resources, 50*(2), 317-372. https://doi.org/10.3368/jhr.50.2.317

Darlington, R. B., & Hayes, A. F. (2017). *Regression analysis and linear models: Concepts, applications, and implementation*. The Guilford Press.

Cribari-Neto, F. (2004). Asymptotic inference under heteroskedasticity of unknown form. *Computational Statistics & Data Analysis, 45*, 215-233. https://doi.org/10.1016/S0167-9473(02)00366-3

Cribari-Neto, F., & Lima, M. G. (2014). New heteroskedasticity-robust standard errors for the linear regression model. *Brazilian Journal of Probability and Statistics, 28*, 83-95.

Cribari-Neto, F., Souza, T., & Vasconcellos, K. L. P. (2007). Inference under heteroskedasticity and leveraged data. *Communications in Statistics - Theory and Methods, 36*, 1877-1888. https://doi.org/10.1080/03610920601126589

Hayes, A.F, & Cai, L. (2007). Using heteroscedasticity-consistent standard error estimators in OLS regression: An introduction and software implementation. *Behavior Research Methods, 39*, 709-722. https://doi.org/10.3758/BF03192961

Huang, F. L., & Li, X. (2022). Using cluster-robust standard errors when analyzing group-randomized trials with few clusters. *Behavior Research Methods, 54*(3), 1181–1199. https://doi.org/10.3758/s13428-021-01627-0

Kuznetsova, A, Brockhoff, P. B., & Christensen, R. H. B. (2017). lmerTest Package: Tests in linear mixed effects models. *Journal of Statistical Software, 82* 13, 1-26. https://doi.org/10.18637/jss.v082.i13.

Imbens, G. W., & Kolesar, M. (2016). Robust standard errors in small samples: Some practical advice. *Review of Economics and Statistics, 98*(4), 701-712. https://doi.org/10.1162/REST_a_00552

Liang, K.-Y., & Zeger, S. L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika, 73*(1), 13-22. https://doi.org/10.1093/biomet/73.1.13

Long, J.S., & Ervin, L.H. (2000). Using heteroscedasticity consistent standard errors in the linear regression model. *The American Statistician, 54*, 217-224. https://doi.org/10.1080/00031305.2000.10474549

Maas, C., & Hox, J. J. (2004). The influence of violations of assumptions on multilevel parameter estimates and their standard errors. *Computational Statistics & Data Analysis, 46*(3), 427-440. https://doi.org/10.1016/j.csda.2003.08.006

MacKinnon, J. G. (2012). Thirty years of heteroskedasticity-robust inference. In X. Chen & N. R. Swanson (Eds.), *Recent advances and future directions in causality, prediction, and specification analysis: Essays in honor of Halbert L. White Jr* (pp. 437-461). Springer. https://doi.org/10.1007/978-1-4614-1653-1_17

Ng, M., & Wilcoy, R. R. (2009). Level robust methods based on the least squares regression estimator. *Journal of Modern Applied Statistical Methods, 8*, 284-395. https://doi.org/10.22237/jmasm/1257033840

Pustejovsky, J. E. & Tipton, E. (2018). Small sample methods for cluster-robust variance estimation and hypothesis testing in fixed effects models. *Journal of Business and Economic Statistics, 36*(4), 672-683. https://doi.org/10.1080/07350015.2016.1247004

Rosopa, P. J., Schaffer, M. M., & Schroeder, A. N. (2013). Managing heteroscedasticity in general linear models. *Psychological Methods, 18*(3), 335-351. https://doi.org/10.1037/a0032553

White, H. (1980). A heteroskedastic-consistent covariance matrix estimator and a direct test of heteroskedasticity. *Econometrica, 48*, 817-838. https://doi.org/10.2307/1912934

Zeileis, A., & Hothorn, T. (2002). Diagnostic checking in regression relationships. *R News, 2*(3), 7-10. http://CRAN.R-project.org/doc/Rnews/

Zeileis A, Köll S, & Graham N (2020). Various versatile variances: An object-oriented implementation of clustered covariances in R. *Journal of Statistical Software, 95*(1), 1-36. https://doi.org/10.18637/jss.v095.i01

Zhang, Y., & Lai, M. H. C. (2024). Evaluating two small-sample corrections for fixed-effects standard errors and inferences in multilevel models with heteroscedastic, unbalanced, clustered data. *Behavior research methods, 56*(6), 5930–5946. https://doi.org/10.3758/s13428-023-02325-9

See Also

coeff.std, write.result

Examples

```
#————————————————————————————————————————————————————————————————————————
# Example 1: Linear model

mod.lm <- lm(mpg ~ cyl + disp, data = mtcars)
coeff.robust(mod.lm)

#————————————————————————————————————————————————————————————————————————
# Example 2: Generalized linear model

mod.glm <- glm(carb ~ cyl + disp, data = mtcars, family = poisson())
coeff.robust(mod.glm)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Example 3: Multilevel and Linear Mixed-Effects Model

# Load lme4 and misty package
misty::libraries(lme4,  nlme, misty)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Cluster-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Grand-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, w1, type = "CGM", cluster = "cluster")

# Estimate two-level mixed-effects model using the lme4 package
mod.lmer <- lmer(y1 ~ x2.c + w1.c + x2.c:w1.c + (1 + x2.c | cluster), data = Demo.twolevel)

# Statistical significance testing based on cluster-robust standard errors
coeff.robust(mod.lmer)

# Estimate two-level mixed-effects model using the nlme package
mod.lme <- lme(y1 ~ x2.c + w1.c + x2.c:w1.c, random = ~ 1 + x2.c | cluster, data = Demo.twolevel)
```

```
# Statistical significance testing based on cluster-robust SE
coeff.robust(mod.lme)

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 3a: Write results into a text file
coeff.robust(mod.lm, write = "Robust_Coef.txt", output = FALSE)

# Example 3b: Write results into an Excel file
coeff.robust(mod.lm, write = "Robust_Coef.xlsx", output = FALSE)

## End(Not run)
```

`coeff.std` *Standardized Coefficients for Linear, Multilevel and Mixed-Effects Models*

Description

This function computes standardized coefficients for linear models estimated by using the `lm()` function and for multilevel and linear mixed-effects models estimated by using the `lmer()` or `lme()` function from the **lme4** or **nlme** package.

Usage

```
coeff.std(model, print = c("all", "stdx", "stdy", "stdyx"), digits = 2, p.digits = 3,
          write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted model of class `"lm"`, `"lmerMod"`, `"lmerModLmerTest"` or `"lme"`.
`print`	a character vector indicating which results to print, i.e. `"all"`, for all results, `"stdx"` for standardizing only the predictor, `"stdy"` for for standardizing only the criterion, and `"stdyx"` for for standardizing both the predictor and the criterion. Note that the default setting is depending on the level of measurement of the predictors, i.e., if all predictors are continuous, the default setting is `print = "stdyx"`; if all predictors are binary, the default setting is `print = "stdy"`, and if predictors are continuous and binary, the default setting is `print = c("stdy", "stdyx")`.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.

`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Linear Regression Model The linear regression model is expressed as follows:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where y_i is the outcome variable for individual i, β_0 is the intercept, β_1 is the slope (aka regression coefficient), x_i is the predictor for individual i, and ϵ_i is the residual for individual i.

The slope β_1 estimated by using the `lm()` function can be standardized with respect to only x, only y, or both y and x:

- **StdX Standardization**: $StdX(\beta_1)$ standardizes with respect to x only and is interpreted as expected difference in y between individuals that differ one standard deviation referred to as $SD(x)$:

$$StdX(\beta_1) = \beta_1 SD(x)$$

- **StdY Standardization**: $StdY(\beta_1)$ standardizes with respect to y only and is interpreted as expected difference in y standard deviation units, referred to as $SD(y)$, between individuals that differ one unit in x:

$$StdY(\beta_1) = \frac{\beta_1}{SD(x)}$$

- **StdYX Standardization**: $StdYX(\beta_1)$ standardizes with respect to both y and x and is interpreted as expected difference in y standard deviation units between individuals that differ one standard deviation in x:

$$StdYX(\beta_1) = \beta_1 \frac{SD(x)}{SD(y)}$$

Note that the $StdYX(\beta_1)$ and the $StdY(\beta_1)$ standardizations are not suitable for the slope of a binary predictor because a one standard deviation change in a binary variable is generally not of interest (Muthen et al, 2016). Accordingly, the function does not provide the $StdYX(\beta_1)$ and the $StdY(\beta_1)$ standardizations whenever a binary vector, factor, or character vector is specified for the predictor variable.

Moderated Regression Model The moderated regression model is expressed as follows:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{1i} x_{2i} + \epsilon_i$$

where β_3 is the slope for the interaction variable $x_1 x_2$.

The slope β_3 is standardized by using the product of standard deviations $SD(x_1)SD(x_2)$ rather than the standard deviation of the product $SD(x_1x_2)$ for the interaction variable x_1x_2 as discussed in Wen et al. (2010).

Note that the function does not use binary variables in the interaction term in standardizing the interaction variable. For example, when standardizing the interaction term $x1 : x2 : x3$ with $x2$ being binary, the product $SD(x_1)SD(x_3)$ while excluding binary predictor `x2` is used to standardize the interaction term.

Polynomial Regression Model The polynomial regression model is expressed as follows:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

where β_2 is the slope for the quadratic term x^2.

The slope β_3 is standardized by using the product of standard deviations $SD(x)SD(x)$ rather than the standard deviation of the product $SD(xx)$ for the quadratic term x^2.

Multilevel and Mixed-Effects Model The random intercept and slope model in the multiple-equation notation is expressed as follows:

- Level 1:

$$y_{ij} = \beta_{0j} + \beta_{1j} x_{ij} + r_{ij}$$

- Level 2:

$$\beta_{0j} = \gamma_{00} + \gamma_{01} z_j + u_{0j}$$
$$\beta_{1j} = \gamma_{10} + u_{1j}$$

The model expressed in the single-equation notation is as follows:

$$y_{ij} = \gamma_{00} + \gamma_{10} x_{ij} + \gamma_{01} z_j + u_{0j} + u_{1j} x_{ij} + r_{ij}$$

where y_{ij} is the outcome variable for individual i in group j, γ_{00} is the fixed-effect average intercept, γ_{10} is the fixed-effect average slope for the Level-1 predictor x, and γ_{01} is the fixed-effect slope for the Level-2 predictor z.

The slopes γ_{10} and γ_{01} are standardized according to the within- and between-group or within- and between-person standard deviations, i.e., slopes are standardizes with respect to the x and y standard deviation relevant for the level of the fixed effect of interest. The resulting standardized slopes are called pseudo-standardized coefficients (Hoffman 2015, p. 342). The StdYX Standardization for γ_{10} and γ_{10} is expressed as follows:

- Level-1 Predictor:

$$StdYX(\gamma_{10}) = \gamma_{10} \frac{SD(x_{ij})}{SD(y_{ij})}$$

- Level-2 Predictor:

$$StdYX(\gamma_{01}) = \gamma_{01} \frac{SD(x_j)}{SD(y_j)}$$

where $SD(x_{ij})$ and $SD(x_j)$ are the standard deviations of the predictors at each analytic level, $SD(y_{ij})$ is the square root of the Level-1 residual variance σ_r^2 and $SD(y_j)$ is square root of the Level-2 intercept variance $\sigma_{u_0}^2$ which are estimated in a null model using the `lmer` function in the **lme4** package using the restricted maximum likelihood estimation method.

The function uses the square root of the Level-1 residual variance σ_r^2 to standardize the slope of the cross-level interaction though it should be noted that it is unclear whether this is the correct approach to standardize the slope of the cross-level interaction.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame with variables used in the analysis
`model`	model specified in `model`
`args`	specification of function arguments
`result`	list with result tables, i.e., `coef` for the regression table including standardized coefficients and `sd` for the standard deviation of the outcome and predictor(s)

Author(s)

Takuya Yanagida

References

Hoffman, L. (2015). *Longitudinal Analysis: Modeling Within-Person Fluctuation and Change*. Routledge.

Muthen, B. O., Muthen, L. K., & Asparouhov, T. (2016). *Regression and mediation analysis using Mplus*. Muthen & Muthen.

Wen, Z., Marsh, H. W., & Hau, K.-T. (2010). Structural equation models of latent interactions: An appropriate standardized solution and its scale-free properties. *Structural Equation Modeling: A Multidisciplinary Journal, 17*, 1-22. https://doi.org/10.1080/10705510903438872

Examples

```
#----------------------------------------------------------------------------
# Linear Model

# Example 1a: Continuous predictors
mod.lm1 <- lm(mpg ~ cyl + disp, data = mtcars)
coeff.std(mod.lm1)

# Example 1b: Print all standardized coefficients
coeff.std(mod.lm1, print = "all")

# Example 1c: Binary predictor
mod.lm2 <- lm(mpg ~ vs, data = mtcars)
coeff.std(mod.lm2)

# Example 1d: Continuous and binary predictors
mod.lm3 <- lm(mpg ~ disp + vs, data = mtcars)
coeff.std(mod.lm3)

# Example 1e: Continuous predictors with interaction term
mod.lm4 <- lm(mpg ~ cyl*disp, data = mtcars)
coeff.std(mod.lm4)
```

```
# Example 1f: Continuous and binary predictor with interaction term
mod.lm5 <- lm(mpg ~ cyl*vs, data = mtcars)
coeff.std(mod.lm5)

# Example 1g: Continuous predictor with a quadratic term
mod.lm6 <- lm(mpg ~ cyl + I(cyl^2), data = mtcars)
coeff.std(mod.lm6)

#-----------------------------------------------------------------------
# Multilevel and Linear Mixed-Effects Model

# Load lme4, nlme, and ggplot2 package
misty::libraries(lme4, nlme)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Cluster-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Grand-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, w1, type = "CGM", cluster = "cluster")

# Estimate models using the lme4 package
mod1a <- lmer(y1 ~ x2.c + w1.c + (1 + x2.c | cluster), data = Demo.twolevel,
              REML = FALSE)
mod2a <- lmer(y1 ~ x2.c + w1.c + x2.c:w1.c + (1 + x2.c | cluster),
              data = Demo.twolevel, REML = FALSE)

# Estimate models using the nlme package
mod1b <- lme(y1 ~ x2.c + w1.c, random = ~ 1 + x2.c | cluster, data = Demo.twolevel,
             method = "ML")
mod2b <- lme(y1 ~ x2.c + w1.c + x2.c:w1.c, random = ~ 1 + x2.c | cluster,
             data = Demo.twolevel, method = "ML")

# Example 2a: Continuous predictors
coeff.std(mod1a)
coeff.std(mod1b)

# Example 2b: Continuous predictors with cross-level interaction
coeff.std(mod2a)
coeff.std(mod2b)

## Not run:
#-----------------------------------------------------------------------
# Example 3: Write Results into a text or Excel file

# Example 3a: Text file
coeff.std(mod.lm1, write = "Std_Coef.txt", output = FALSE)

# Example 3b: Excel file
coeff.std(mod.lm1, write = "Std_Coef.xlsx", output = FALSE)
```

```
## End(Not run)
```

---

cohens.d *Cohen's d*

Description

This function computes Cohen's d for one-sample, two-sample (i.e., between-subject design), and paired-sample designs (i.e., within-subject design) for one or more variables, optionally by a grouping and/or split variable. In a two-sample design, the function computes the standardized mean difference by dividing the difference between means of the two groups of observations by the weighted pooled standard deviation (i.e., Cohen's d_s according to Lakens, 2013) by default. In a paired-sample design, the function computes the standardized mean difference by dividing the mean of the difference scores by the standard deviation of the difference scores (i.e., Cohen's d_z according to Lakens, 2013) by default. Note that by default Cohen's d is computed without applying the correction factor for removing the small sample bias (i.e., Hedges' g).

Usage

```
cohens.d(...)

## Default S3 method:
cohens.d(data = NULL, ..., x = NULL, y = NULL, mu = 0, paired = FALSE,
         weighted = TRUE, cor = TRUE, ref = NULL, correct = FALSE,
         alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
         group = NULL, split = NULL, sort.var = FALSE, digits = 2, as.na = NULL,
         write = NULL, append = TRUE, check = TRUE, output = TRUE)

## S3 method for class 'formula'
cohens.d(formula, data, weighted = TRUE, cor = TRUE, ref = NULL,
         correct = FALSE, alternative = c("two.sided", "less", "greater"),
         conf.level = 0.95, group = NULL, split = NULL, sort.var = FALSE,
         na.omit = FALSE, digits = 2, as.na = NULL, write = NULL, append = TRUE,
         check = TRUE, output = TRUE, ...)
```

Arguments

`data`	a data frame. Note that data are specified either by using the argument `data` or by using the arguments `x` and `y`.
`...`	an expression indicating the variable names in `data`, e.g., `cohens.d(dat, x1, x2)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`x`	a numeric vector of data values.
`y`	a numeric vector of data values.

formula	a formula of the form y ~ group for one outcome variable or cbind(y1, y2, y3) ~ group for more than one outcome variable where y is a numeric variable giving the data values and group a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
mu	a numeric value indicating the reference mean.
paired	logical: if TRUE, Cohen's d for a paired-sample design is computed.
weighted	logical: if TRUE (default), the weighted pooled standard deviation is used to compute the standardized mean difference between two groups of a two-sample design (i.e., paired = FALSE), while standard deviation of the difference scores is used to compute the standardized mean difference in a paired-sample design (i.e., paired = TRUE).
cor	logical: if TRUE (default), paired = TRUE, and weighted = FALSE, Cohen's d for a paired-sample design while controlling for the correlation between the two sets of measurement is computed. Note that this argument is only used in a paired-sample design (i.e., paired = TRUE) when specifying weighted = FALSE.
ref	character string "x" or "y" for specifying the reference reference group when using the default cohens.d() function or a numeric value or character string indicating the reference group in a two-sample design when using the formula cohens.d() function. The standard deviation of the reference variable or reference group is used to standardized the mean difference. Note that this argument is only used in a two-sample design (i.e., paired = FALSE).
correct	logical: if TRUE, correction factor to remove positive bias in small samples is used.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	a numeric vector, character vector or factor as grouping variable.
split	a numeric vector, character vector or factor as split variable.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
digits	an integer value indicating the number of decimal places to be used for displaying results.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to y but not to group in a two-sample design, while as.na() function is applied to pre and post in a paired-sample design.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Cohen (1988, p.67) proposed to compute the standardized mean difference in a two-sample design by dividing the mean difference by the unweighted pooled standard deviation (i.e., `weighted = FALSE`).

Glass et al. (1981, p. 29) suggested to use the standard deviation of the control group (e.g., `ref = 0` if the control group is coded with 0) to compute the standardized mean difference in a two-sample design (i.e., Glass's Δ) since the standard deviation of the control group is unaffected by the treatment and will therefore more closely reflect the population standard deviation.

Hedges (1981, p. 110) recommended to weight each group's standard deviation by its sample size resulting in a weighted and pooled standard deviation (i.e., `weighted = TRUE`, default). According to Hedges and Olkin (1985, p. 81), the standardized mean difference based on the weighted and pooled standard deviation has a positive small sample bias, i.e., standardized mean difference is overestimated in small samples (i.e., sample size less than 20 or less than 10 in each group). However, a correction factor can be applied to remove the small sample bias (i.e., `correct = TRUE`). Note that the function uses a gamma function for computing the correction factor, while a approximation method is used if computation based on the gamma function fails.

Note that the terminology is inconsistent because the standardized mean difference based on the weighted and pooled standard deviation is usually called Cohen's d, but sometimes called Hedges' g. Oftentimes, Cohen's d is called Hedges' d as soon as the small sample correction factor is applied. Cumming and Calin-Jageman (2017, p.171) recommended to avoid the term Hedges' g , but to report which standard deviation was used to standardized the mean difference (e.g., unweighted/weighted pooled standard deviation, or the standard deviation of the control group) and whether a small sample correction factor was applied.

As for the terminology according to Lakens (2013), in a two-sample design (i.e., `paired = FALSE`) Cohen's d_s is computed when using `weighted = TRUE` (default) and Hedges's g_s is computed when using `correct = TRUE` in addition. In a paired-sample design (i.e., `paired = TRUE`), Cohen's d_z is computed when using `weighted = TRUE, default`, while Cohen's d_{rm} is computed when using `weighted = FALSE` and `cor = TRUE, default` and Cohen's d_{av} is computed when using `weighted = FALSE` and `cor = FALSE`. Corresponding Hedges' g_z, g_{rm}, and g_{av} are computed when using `correct = TRUE` in addition.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`sample`	type of sample, i.e., one-, two-, or, paired-sample
`data`	matrix or data frame specified in `x`
`args`	specification of function arguments
`result`	result table

Author(s)

Takuya Yanagida

References

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Academic Press.

Cumming, G., & Calin-Jageman, R. (2017). *Introduction to the new statistics: Estimation, open science, & beyond*. Routledge.

Glass. G. V., McGaw, B., & Smith, M. L. (1981). *Meta-analysis in social research*. Sage Publication.

Goulet-Pelletier, J.-C., & Cousineau, D. (2018) A review of effect sizes and their confidence intervals, Part I: The Cohen's d family. *The Quantitative Methods for Psychology, 14*, 242-265. https://doi.org/10.20982/tqmp.14.4.p242

Hedges, L. V. (1981). Distribution theory for Glass's estimator of effect size and related estimators. *Journal of Educational Statistics, 6*(3), 106-128.

Hedges, L. V. & Olkin, I. (1985). *Statistical methods for meta-analysis*. Academic Press.

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs. *Frontiers in Psychology, 4*, 1-12. https://doi.org/10.3389/fpsyg.2013.00863

See Also

test.t, test.z, effsize, cor.matrix, na.auxiliary

Examples

```
#————————————————————————————————————————————————————————————————————
# One-Sample Design

# Example 1a: Cohen's d.z with two-sided 95% CI, population mean = 3
cohens.d(mtcars, mpg, mu = 20)

# Example 1b: Alternative specification without using the '...' argument
cohens.d(mtcars$mpg, mu = 20)

# Example 1c: Cohen's d.z (aka Hedges' g.z) population mean = 3,
# with small sample correction factor
cohens.d(mtcars, mpg, mu = 20, correct = TRUE)

# Example 1d: Cohen's d.z, population mean = 3 by 'vs' separately
cohens.d(mtcars, mpg, mu = 20, group = "vs")

# Example 1e: Cohen's d.z, population mean = 20, split analysis by 'vs'
cohens.d(mtcars, mpg, mu = 20, split = "vs")

# Example 1f: Cohen's d.z, population mean = 3, by 'vs' separately, split by 'am'
cohens.d(mtcars, mpg, mu = 20, group = "vs", split = "am")

#————————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 2a: Cohen's d.s with two-sided 95% CI, weighted pooled SD
cohens.d(mpg ~ vs, data = mtcars)
```

```
# Example 2b: Alternative specification without using the '...' argument
test.t(x = df.subset(mtcars, mpg, subset = vs == 0),
       y = df.subset(mtcars, mpg, subset = vs == 1))

# Example 2c: Cohen's d.s with one-sided 99% CI
cohens.d(mpg ~ vs, data = mtcars, alternative = "greater", conf.level = 0.99)

# Example 2d: Cohen's d.s for more than one variable
cohens.d(cbind(mpg, disp, hp) ~ vs, data = mtcars)

# Example 2e: Cohen's d, unweighted SD
cohens.d(mpg ~ vs, data = mtcars, weighted = FALSE)

# Example 2f: Cohen's d.s (aka Hedges' g.s), weighted pooled SD,
#             with small sample correction factor
cohens.d(mpg ~ vs, data = mtcars, correct = TRUE)

# Example 2g: Cohen's d (aka Hedges' g), unweighted SD,
#             with small sample correction factor
cohens.d(mpg ~ vs, data = mtcars, weighted = FALSE, correct = TRUE)

# Example 2h: Cohen's d (aka Glass's delta), SD of reference group 1
cohens.d(mpg ~ vs, data = mtcars, ref = 0)

# Example 2i: Cohen's d.s,  by 'am' separately
cohens.d(mpg ~ vs, data = mtcars, group = "am")

# Example 2j: Cohen's d.s, split analysis by 'am'
cohens.d(mpg ~ vs, data = mtcars, split = "am")

#------------------------------------------------------------------------
# Paired-Sample Design

# Example 3a: Cohen's d.z with two-sided 95% CI
# SD of the difference scores
cohens.d(mtcars, drat, wt, paired = TRUE)

# Example 3b:  Alternative specification without using the '...' argument
cohens.d(mtcars[, c("drat", "wt")], paired = TRUE)

# Example 3c: Alternative specification without using the '...' argument
cohens.d(x = mtcars$drat, y = mtcars$wt, paired = TRUE)

# Example 3d: Cohen's d.rm, controlling for the correlation between measures
cohens.d(mtcars, drat, wt, paired = TRUE, weighted = FALSE)

# Example 3e: Cohen's d.av, without controlling for the correlation between measures
cohens.d(mtcars, drat, wt, paired = TRUE, weighted = FALSE, cor = FALSE)

# Example 3f: Cohen's d.z (aka Hedges' g.z),# SD of the differnece scores
cohens.d(mtcars, drat, wt, paired = TRUE, correct = TRUE)

# Example 3g: Cohen's d.z with two-sided 95% CI
```

```
#             SD of the difference scores, by 'vs' separately
cohens.d(mtcars, drat, wt, paired = TRUE, group = "vs")

# Example 3h:  Cohen's d.z with two-sided 95% CI
#              SD of the difference scores, split analysis by 'vs'
cohens.d(mtcars, drat, wt, paired = TRUE, split = "vs")
```

`cor.matrix` *Correlation Matrix*

Description

This function computes a correlation matrix based on Pearson product-moment correlation coefficient, Spearman's rank-order correlation coefficient, Kendall's Tau-b correlation coefficient, Kendall-Stuart's Tau-c correlation coefficient, tetrachoric correlation coefficient, or polychoric correlation coefficient and computes significance values (p-values) for testing the two-sided hypothesis H0: $\rho = 0$ for all pairs of variables. Statistically significant correlations can be highlighted by specifying the argument `color`.

Usage

```
cor.matrix(data, ...,
        method = c("pearson", "spearman", "kendall-b", "kendall-c", "tetra", "poly"),
            group = NULL, exact = FALSE, continuity = TRUE, ml = FALSE,
            print = c("all", "cor", "n", "stat", "df", "p"),
            tri = c("both", "lower", "upper"), alpha = 0.05,
            color = "default", style = c("regular", "bold", "italic"),
            p.adj = "none", na.omit = FALSE, digits = 2, p.digits = 3,
            as.na = NULL,  plot = FALSE, geom = c("none", "label"),
            filename = NULL, width = NA, height = NA, dpi = 600,
            write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with numeric variables, i.e., factors and character variables are excluded from `data` before conducting the analysis.
`...`	an expression indicating the variable names in `data`, e.g., `cor.matrix(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`method`	a character vector indicating which correlation coefficient is to be computed, i.e. `"pearson"` for Pearson product-moment correlation coefficient (default), `"spearman"` for Spearman's rank-order correlation coefficient, `"kendall-b"` for Kendall's Tau-b correlation coefficient, `"kendall-c"` for Kendall-Stuart's Tau-c correlation coefficient, `"tetra"` for tetrachoric correlation coefficient, and `"poly"` for polychoric correlation coefficient.
`group`	either a character string indicating the variable name of the grouping variable in `data`, or a vector representing the grouping variable. Note that the grouping variable is limited to two groups.

`exact` logical: if `TRUE` (default), an exact p-value is computed for Spearman's rank-order correlation coefficient and Kendall's Kendall's Tau-b correlation coefficient. Note that the exact p-value is not available in the presence of ties.

`continuity` logical: if `TRUE` (default), continuity correction is used for testing Spearman's rank-order correlation coefficient and Kendall's Tau-b correlation.

`ml` logical: if `FALSE` (default), a two-step approximation is used to compute the tetrachoric and polychoric correlation coefficient, while the maximum-likelihood (ML) estimate is computed if `TRUE`. Note that ML estimation is computationally expensive, i.e., takes a lot of time.

`print` a character vector indicating which results to show on the console, i.e. `"all"` for all results, `"cor"` for correlation coefficients, `"n"` for the sample sizes, `"stat"` for the test statistic, `"df"` for the degrees of freedom, and `"p"` for *p*-values.

`tri` a character string indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.

`alpha` a numeric value between 0 and 1 indicating the significance level at which correlation coefficients are are highlighted when specifying the argument `color`.

`color` a character string indicating the text color for highlighting statistically significant correlation coefficients , i.e., `"default"` (default) for the default text color without color coding and various text colors for highlighting like `"red"`, `"b.red"`, `"green"`, `"b.green"`, `"blue"`, or `"b.blue"`, see the help page of the `chr.color` function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to `"default"` when using R Markdown and when writing the output into a text file (`.txt`).

`style` a character vector indicating the font style for statistically significant correlation coefficients, i.e., `"regular"` (default) for regular text, `"bold"` for bold text, and `"italic"` for italic text. Note that the font style `"bold"` and `"italic"` can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument `color` needs to be specified to change the style of the text, e.g. `color = "black"` and `style = "bold"` to for bold text.

`p.adj` a character string indicating an adjustment method for multiple testing based on `p.adjust`, i.e., `none` (default), `bonferroni`, `holm`, `hochberg`, `hommel`, `BH`, `BY`, or `fdr`. See help page of the `p.adjust` function.

`na.omit` logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion); if `FALSE` (default), pairwise deletion is used.

`digits` an integer value indicating the number of decimal places to be used for displaying correlation coefficients.

`p.digits` an integer value indicating the number of decimal places to be used for displaying *p*-values.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.

`plot` logical: if `TRUE`,a correlation heat map is drawn. For additional plotting arguments, see `Details` in the help page of the function `plot.misty.object`.

geom	character vector indicating which plot elements to draw, i.e., "none" for not printing correlation coefficients inside every tile, "label" for printing correlation coefficients inside every tile.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame used for the current analysis
args	specification of function arguments
result	list with result tables, i.e., cor for the correlation matrix, n for a matrix with the sample sizes, stat for a matrix with the test statistics, df for a matrix with the degrees of freedom, and p-value for the matrix with the significance values (*p*-values)

Note

This function uses a modified copy of the polychor() function in the **polycor** package by John Fox when requesting tetrachoric or polychoric correlation coefficients.

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Revelle, W. (2018) *psych: Procedures for personality and psychological research*. Northwestern University, Evanston, Illinois, USA, https://CRAN.R-project.org/package=psych Version = 1.8.12.

See Also

write.result, cohens.d, effsize, multilevel.cor, multilevel.icc, na.auxiliary

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Pearson Product-Moment Correlation Coefficient

# Example 1a: Pearson product-moment correlation matrix using pairwise deletion
cor.matrix(airquality, Ozone:Wind)

# Example 1b:  Alternative specification without using the '...' argument
cor.matrix(airquality[, c("Ozone", "Solar.R", "Wind")])

# Example 2a: Highlight statistically significant result in bright red
cor.matrix(airquality, Ozone, Solar.R, Wind, color = "b.red")

# Example 2b: Highlight statistically significant result in boldface
cor.matrix(airquality, Ozone, Solar.R, Wind, color = "black", style = "bold")

# Example 3a: Print sample size, degrees of freedom, and significance values
cor.matrix(airquality, Ozone, Solar.R, Wind, print = "all")

# Example 3b: Listwise deletion
cor.matrix(airquality, Ozone, Solar.R, Wind, na.omit = TRUE)

# Example 3c: Significance values with Bonferroni correction
cor.matrix(airquality, Ozone, Solar.R, Wind, print = "all", p.adj = "bonferroni")

#————————————————————————————————————————————————————————————————————————
# Spearman's Rank-Order Correlation Coefficient and Kendall's Tau

# Example 4a: Spearman's rank-order correlation matrix
cor.matrix(airquality, Ozone, Solar.R, Wind, method = "spearman")

# Example 4b: Kendall's Tau-c
cor.matrix(airquality, Ozone, Solar.R, Wind, method = "kendall-c")

#————————————————————————————————————————————————————————————————————————
# Tetrachoric and Polychoric Correlation Coefficient

# Example 5a: Tetrachoric correlation matrix
```

```
cor.matrix(data.items, +ditem, method = "tetra")

# Example 5b: Polychoric correlation matrix
cor.matrix(data.items, +pitem, method = "poly")

#————————————————————————————————————————————————————————————————————————
# Grouping Variable

# Example 6a: Results for group 'vs' = "0" and "1" separately
cor.matrix(mtcars, mpg:disp, group = "vs")

# Example 6b: Alternative specification without using the '...' argument
cor.matrix(mtcars[, c("mpg", "cyl", "disp")], group = mtcars$vs)

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 7a: Correlation heat map
cor.matrix(mtcars, plot = TRUE)

# Example 7b: Correlation heat map or group 'vs' = "0" and "1" separately
cor.matrix(mtcars, group = "vs", plot = TRUE)

# Example 7c: Highlight statistically significant result in blue
cor.matrix(mtcars, color = "blue", plot = TRUE)

# Example 7d: Highlight statistically significant result in boldface
cor.matrix(mtcars, color = "black", style = "bold", plot = TRUE)

# Example 7e: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- cor.matrix(mtcars)
plot(object, scale.fill.low = "#DC267F", scale.fill.high = "#009E73",
     text.col = c("gray80", "red"), text.size = 4.5, axis.text.size = 13)

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 8a: Write Results into a text file
cor.matrix(airquality, Ozone, Solar.R, Wind, print = "all", write = "Correlation.txt")

# Example 8b: Write Results into an Excel file
cor.matrix(airquality, Ozone, Solar.R, Wind, print = "all", write = "Correlation.xlsx")

# Example 8c: Save plot as PNG fine
cor.matrix(mtcars, plot = TRUE, filename = "Correlation.png", width = 8, height = 7)

## End(Not run)
```

---

crosstab *Cross Tabulation*

Description

This function creates a two-way and three-way cross tabulation with absolute frequencies and row-wise, column-wise and total percentages.

Usage

```
crosstab(data, ..., print = c("no", "all", "row", "col", "total"),
         freq = TRUE, split = FALSE, na.omit = TRUE, digits = 2, as.na = NULL,
         write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with two or three columns.
`...`	an expression indicating the variable names in `data`, e.g., `crosstab(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`print`	a character vector indicating which percentage(s) to be printed on the console, i.e., no percentages (`"no"`) (default), all percentages (`"all"`), row-wise percentages (`"row"`), column-wise percentages (`"col"`), and total percentages (`"total"`).
`freq`	logical: if `TRUE` (default), absolute frequencies will be included in the cross tabulation.
`split`	logical: if `TRUE`, output table is split in absolute frequencies and percentage(s).
`na.omit`	logical: if `TRUE` (default), incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`digits`	an integer indicating the number of decimal places digits to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is printed on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data`

`args`	specification of function arguments
`result`	list with result tables, i.e., `crosstab` for the cross tabulation, `freq.a` for the absolute frequencies, `perc.r` for the row-wise percentages, `perc.c` for the column-wise percentages, `perc.t` for the total percentages

Author(s)

Takuya Yanagida

`write.result`, `freq`, `descript`, `multilevel.descript`, `na.descript`.

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Examples

```
#————————————————————————————————————————————————————————————————————————
# Two-Dimensional Table

# Example 1a: Cross Tabulation for 'vs' and 'am'
crosstab(mtcars, vs, am)

# Alternative specification without using the '...' argument
crosstab(mtcars[, c("vs", "am")])

# Example 1b: Cross Tabulation, print all percentages
crosstab(mtcars, vs, am, print = "all")

# Example 1c: Cross Tabulation, print row-wise percentages
crosstab(mtcars, vs, am, print = "row")

# Example 1d: Cross Tabulation, print col-wise percentages
crosstab(mtcars, vs, am, print = "col")

# Example 1e: Cross Tabulation, print all percentages, split output table
crosstab(mtcars, vs, am, print = "all", split = TRUE)

#————————————————————————————————————————————————————————————————————————
# Three-Dimensional Table

# Example 2a: Cross Tabulation for 'vs', 'am', ane 'gear'
crosstab(mtcars, vs:gear)

# Alternative specification without using the '...' argument
crosstab(mtcars[, c("vs", "am", "gear")])

# Example 2b: Cross Tabulation, print all percentages
crosstab(mtcars, vs:gear, print = "all")

# Example 2c: Cross Tabulation, print all percentages, split output table
```

```
crosstab(mtcars, vs:gear, print = "all", split = TRUE)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 3a: Write Results into a text file
crosstab(mtcars, vs:gear, print = "all", write = "Crosstab.txt")

# Example 3b: Write Results into an Excel file
crosstab(mtcars, vs:gear, print = "all", write = "Crosstab.xlsx")

## End(Not run)
```

`data.items` *Multiple-Choice, Dichotomous, and Polytomous Item Data*

Description

An example data set with ten multiple-choice test items, scored dichotomous test items, and six polytomous item with four response options, i.e., 0 = "I strongly disagree", 1 = "I disagree", 2 = "I agree", and 3 = "I strongly agree". Note that the inverted items `pitem2` and `pitem3` need to be reverse coded before conducting the analysis. The example data is based on the `dichotomous` and `timms2011_usa` data objects provided in the package **itemanalysis** by Cengiz Zopluoglu.

Format

A data frame with 1,000 rows and 27 columns:

id Identifier, i.e., consecutive numbers from 1 to 1,000

group2 arbitrary grouping variable with two groups

group4 arbitrary grouping variable with fout groups

mitem1 MC test items 1 with four response options A, B, and C, and D

mitem2 MC test items 2 with four response options A, B, C, and D

mitem3 MC test items 3 with three response options A, B, and C

mitem4 MC test items 4 with three response options A, B, and C

mitem5 MC test items 5 with four response options A, B, C, and D

mitem6 MC test items 6 with three response options A, B, and C

mitem7 MC test items 7 with four response options A, B, C, and D

mitem8 MC test items 8 with four response options A, B, C, and D

mitem9 MC test items 9 with three response options A, B, and C

mitem10 MC test items 10 with three response options A, B, and C

ditem1 Scored MC test item 1

ditem2 Scored MC test item 2

ditem3 Scored MC test item 3

ditem4 Scored MC test item 4

ditem5 Scored MC test item 5

ditem6 Scored MC test item 6

ditem7 Scored MC test item 7

ditem8 Scored MC test item 8

ditem9 Scored MC test item 9

ditem10 Scored MC test item 10

pitem1 I enjoy learning mathematics.

pitem2 I wish have not to study Math. (inverted item)

pitem3 Mathematics is boring.(inverted item)

pitem4 I learn interesting things in mathematics class.

pitem5 I like mathematics.

pitem6 I think it's important to do well in mathematics.

pitem2r Reverse coded item 'pitem2'

pitem3r Reverse coded item 'pitem3'

Source

<https://cran.r-project.org/web/packages/itemanalysis/>

`descript` *Descriptive Statistics*

Description

This function computes summary statistics for one or more than one variable, optionally by a grouping and/or split variable. By default, the function prints the number of observations (`n`), number of missing values (`nNA`), percentage of missing values (`%NA`), number of unique elements after omitting missing values (`nUQ`), arithmetic mean (`M`), standard deviation (`SD`), minimum (`Min`), percentage of observations at the minimum (`%Min`), maximum (`Max`), percentage of observations at the maximum (`%Max`), skewness (`Skew`), and kurtosis (`Kurt`).

Usage

```
descript(data, ...,
         print = c("all", "default", "n", "nNA", "pNA", "nUQ", "m", "se.m",
                "var", "sd", "min", "p.min", "p25", "med", "p75", "max", "p.max",
                   "range", "iqr", "skew", "kurt"),
         group = NULL, split = NULL, sample = FALSE, sort.var = FALSE,
         na.omit = FALSE, digits = 2, as.na = NULL, write = NULL, append = TRUE,
         check = TRUE, output = TRUE)
```

Arguments

data	a numeric vector or data frame with numeric variables, i.e., factors and character variables are excluded from data before conducting the analysis.
...	an expression indicating the variable names in data, e.g., descript(dat, x1, x2, x3). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
print	a character vector indicating which statistical measures to be printed on the console, i.e., n (number of observations), nNA (number of missing values), pNA (percentage of missing values), nUQ (number of unique elements after omitting missing values), m (arithmetic mean), se.m (standard error of the arithmetic mean), var (variance), sd (standard deviation), med (median),min (minimum), p.min (percentage of observations at the minimum), p25 (25th percentile, first quartile), p75 (75th percentile, third quartile), max (maximum), p.max (percentage of observations at the maximum), range (range), iqr (interquartile range), skew (skewness), and kurt (excess kurtosis). The default setting is print = c("n", "nNA", "pNA", "nUQ", "m", "sd", "min", "pmin", "max", "p.max", "skew", "kurt").
group	a numeric vector, character vector or factor as grouping variable. Alternatively, a character string indicating the variable name of the grouping variable in data can be specified.
split	a numeric vector, character vector or factor as split variable. Alternatively, a character string indicating the variable name of the split variable in data can be specified.
sample	logical: if TRUE (default), the univariate sample skewness or kurtosis is computed, while the population skewness or kurtosis is computed when sample = FALSE.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data, but not to group or split.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Floor and Ceiling Effects This function computes the percentage of observations at both the minimum and maximum to evaluate floor and ceiling effects in continuous variables. Historically, floor or ceiling effects are considered to be present if more than 15% of observations are at the lowest or highest possible score (McHorney & Tarlov, 1995; Terwee et al., 2007). Muthen (2023, see video at 7:58) noted the rule of thumb that linear models should be avoided when the floor or ceiling effect of the outcome variable exceeds 25%.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the input specified in `data`, `group`, and `split`
`args`	specification of function arguments
`result`	result table

Author(s)

Takuya Yanagida

References

McHorney, C. A., & Tarlov, A. R. (1995). Individual-patient monitoring in clinical practice: are available health status surveys adequate?. *Quality of Life Research, 4*(4), 293-307. https://doi.org/10.1007/BF01593882

Muthen, B. (2023, Feb. 28). *Mplus Web Talk No. 6 - Using Mplus To Do Dynamic Structural Equation Modeling: Segment 3, Descriptive Analyses* [Video]. YouTube. https://www.statmodel.com/Webtalk6.shtml

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Terwee, C. B., Bot, S. D., de Boer, M. R., van der Windt, D. A., Knol, D. L., Dekker, J., Bouter, L. M., & de Vet, H. C. (2007). Quality criteria were proposed for measurement properties of health status questionnaires. *Journal of Clinical Epidemiology, 60*(1), 34-42. https://doi.org/10.1016/j.jclinepi.2006.03.012

See Also

`ci.mean`, `ci.mean.diff`, `ci.median`, `ci.prop`, `ci.prop.diff`, `ci.var`, `ci.sd`, `freq`, `crosstab`, `multilevel.descript`, `na.descript`.

Examples

```
#————————————————————————————————————————————————————————————————
# Descriptive Statistics

# Example 1a: Descriptive statistics for 'mpg', 'cyl', and 'hp'
descript(mtcars, mpg, cyl, hp)
```

```
# Example 1b: Alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp")])

# Example 1c: Print all available statistical measures
descript(mtcars, mpg, cyl, hp, print = "all")

# Example 1d: Print default statistical measures plus median
descript(mtcars, mpg, cyl, hp, print = c("default", "med"))

#------------------------------------------------------------------------
# Grouping Variable

# Example 2a: Grouping variable
descript(mtcars, mpg, cyl, hp, group = "vs")

# Example 2b: Alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp")], group = mtcars$vs)

# Example 2c: Another alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp", "vs")], group = "vs")

# Example 2d: Sort result table by variables
descript(mtcars, mpg, cyl, hp, group = "vs", sort.var = TRUE)

#------------------------------------------------------------------------
# Split Variable

# Example 3a: Split variable
descript(mtcars, mpg, cyl, hp, split = "am")

# Example 3b: Alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp")], split = mtcars$am)

# Example 3c: Another alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp", "am")], split = "am")

#------------------------------------------------------------------------
# Grouping and Split Variable

# Example 4a: Grouping and split variable
descript(mtcars, mpg, cyl, hp, group = "vs", split = "am")

# Example 4b: Alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp")], group = mtcars$vs, split = mtcars$am)

# Example 4c: Another alternative specification without using the '...' argument
descript(mtcars[, c("mpg", "cyl", "hp", "vs", "am")], group = "vs", split = "am")

# Example 4d: Sort result table by variables
descript(mtcars, mpg, cyl, hp, group = "vs", split = "am", sort.var = TRUE)

## Not run:
```

```
#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 3a: Text file
descript(mtcars, write = "Descript.txt")

# Example 3b: Excel file
descript(mtcars, write = "Descript.xlsx")

## End(Not run)
```

---

df.check *Data Check*

Description

This function is a wrapper around the functions `dim` for the number of rows and columns, `names` for the variable names, `df.head` for the first rows, and `df.tail` for the last rows of a data frame.

Usage

```
df.check(data, print = c("dim", "names", "head", "tail"), n = 4,
         digits = 3, width = 20, row.names = TRUE, row.names.col = "gray2",
         message = TRUE, message.col = "b.blue", check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame.
`print`	a character vector indicating which results to show on the console, i.e., `"dim"`, for the number of rows and number of columns, `"names"` for the variable names, `"head"` for the first rows of the data frame, and `"tail"` for the last rows of the data frame.
`n`	a numeric value indicating the number of rows to be printed on the console.
`digits`	a numeric value indicating the maximum number of decimal places to be used.
`width`	a numeric value indicating the maximum width of the character strings in the vector.
`row.names`	logical: if `TRUE`, row names of the data frame are printed on the console.
`row.names.col`	a character string indicating the text color for the row names, see `color` argument of the `chr.color` function.
`message`	logical: if `TRUE`, number of remaining rows and columns are printed on the console.
`message.col`	a character string indicating the text color for the number of remaining rows and columns printed on the console, see `color` argument of the `chr.color` function.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Note that this function only provides a basic data check suitable for checking a data frame after importing data into R and is not designed to offer a thorough data check (e.g., identifying duplicate IDs or inconsistencies in the data).

Author(s)

Takuya Yanagida

See Also

df.duplicated, df.unique, df.head, df.tail, df.long, df.wide, df.merge, df.move, df.rbind, df.rename, df.sort, df.subset

Examples

```
# Example 1: Check data frame mtcars
df.check(mtcars)
```

---

df.duplicated *Extract Duplicated or Unique Rows*

Description

The function df.duplicated extracts duplicated rows and the function df.unique extracts unique rows from a matrix or data frame.

Usage

```
df.duplicated(data, ..., first = TRUE, keep.all = TRUE, from.last = FALSE,
              keep.row.names = TRUE, check = TRUE)

df.unique(data, ..., keep.all = TRUE, from.last = FALSE,
          keep.row.names = TRUE, check = TRUE)
```

Arguments

data	a data frame.
...	an expression indicating the variable names in data used to determine duplicated or unique rows.e.g., df.duplicated(x1, x2, data = dat). Note that the operators ., +, -, ~, :, ::, and ! can also be used to select variables, see Details in the df.subset function.
first	logical: if TRUE (default), the df.duplicated() function will return duplicated rows including the first of identical rows.
keep.all	logical: if TRUE (default), the function will return all variables in data after extracting duplicated or unique rows based on the variables specified in the argument ....

`from.last`	logical: if `TRUE`, duplication will be considered from the reversed side, i.e., the last of identical rows would correspond to `duplicated = FALSE`. Note that this argument is only used when `first = FALSE`.
`keep.row.names`	logical: if `TRUE` (default), the row names from `data` are kept, otherwise they are set to `NULL`.
`check`	logical: if `TRUE` (default), argument specification is checked.

Details

Note that `df.unique(x)` is equivalent to `unique(x)`. That is, the main difference between the `df.unique()` and the `unique()` function is that the `df.unique()` function provides the `...` argument to specify a variable or multiple variables which are used to determine unique rows.

Value

Returns duplicated or unique rows of the data frame in `...` or `data`.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`df.check`, `df.head`, `df.tail`, `df.long`, `df.wide`, `df.merge`, `df.move`, `df.rbind`, `df.rename`, `df.sort`, `df.subset`

Examples

```
dat <- data.frame(x1 = c(1, 1, 2, 1, 4), x2 = c(1, 1, 2, 1, 6),
                  x3 = c(2, 2, 3, 2, 6), x4 = c(1, 1, 2, 2, 4),
                  x5 = c(1, 1, 4, 4, 3))

#————————————————————————————————————————————————————————————————————————
# df.duplicated() Function

# Example 1a: Extract duplicated rows based on all variables
df.duplicated(dat)

# Example 1b: Extract duplicated rows based on 'x4'
df.duplicated(dat, x4)

# Example 1c: Extract duplicated rows based on 'x2' and 'x3'
df.duplicated(dat, x2, x3)

# Example 1d: Extract duplicated rows based on all variables
# exclude first of identical rows
```

```
df.duplicated(dat, first = FALSE)

# Example 1e: Extract duplicated rows based on 'x2' and 'x3'
# do not return all variables
df.duplicated(dat, x2, x3, keep.all = FALSE)

# Example 1f: Extract duplicated rows based on 'x4'
# consider duplication from the reversed side
df.duplicated(dat, x4, first = FALSE, from.last = TRUE)

# Example 1g: Extract duplicated rows based on 'x2' and 'x3'
# set row names to NULL
df.duplicated(dat, x2, x3, keep.row.names = FALSE)

#————————————————————————————————————————————————————————————————————
# df.unique() Function

# Example 2a: Extract unique rows based on all variables
df.unique(dat)

# Example 2b: Extract unique rows based on 'x4'
df.unique(dat, x4)

# Example 2c: Extract unique rows based on 'x1', 'x2', and 'x3'
df.unique(dat, x1, x2, x3)

# Example 2d: Extract unique rows based on 'x2' and 'x3'
# do not return all variables
df.unique(dat, x2, x3, keep.all = FALSE)

# Example 2e: Extract unique rows based on 'x4'
# consider duplication from the reversed side
df.unique(dat, x4, from.last = TRUE)

# Example 2f: Extract unique rows based on 'x2' and 'x3'
# set row names to NULL
df.unique(dat, x2, x3, keep.row.names = FALSE)
```

`df.head` *Print the First and Last Rows of a Data Frame*

Description

The function `df.head` prints the first rows of a data frame and the function `df.tail` prints the last rows of a data frame and prints as many columns as fit on the console supplemented by a summary of the remaining rows and columns.

Usage

```
df.head(data, n = 6, digits = 3, width = 20, factor.labels = TRUE,
```

```
        row.names = TRUE, row.names.col = "gray2", message = TRUE,
        message.col = "b.blue", check = TRUE, output = TRUE)

df.tail(data, n = 6, digits = 3, width = 20, factor.labels = TRUE,
        row.names = TRUE, row.names.col = "gray2", message = TRUE,
        message.col = "b.blue", check = TRUE, output = TRUE)
```

Arguments

data	a data frame.
n	a numeric value indicating the number of rows to be printed on the console.
digits	a numeric value indicating the maximum number of decimal places to be used.
width	a numeric value indicating the maximum width of the character strings in the vector.
factor.labels	logical: if TRUE, factor labels will be printed on the console.
row.names	logical: if TRUE, row names of the data frame are printed on the console.
row.names.col	a character string indicating the text color for the row names, see color argument of the chr.color function.
message	logical: if TRUE, number of remaining rows and columns are printed on the console.
message.col	a character string indicating the text color for the number of remaining rows and columns printed on the console, see color argument of the chr.color function.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns a list with following entries:

df	data frame specified in data with the first or last n rows of the data frame with as many columns as fit on the console
row.col	character string indicating the remaining rows and columns

Author(s)

Takuya Yanagida

See Also

df.check, df.duplicated, df.unique, df.long, df.wide, df.merge, df.move, df.rbind, df.rename, df.sort, df.subset

Examples

```
# Example 1: Print first and last six rows
df.head(mtcars)
df.tail(mtcars)

# Example 2: Print first and last six rows without row names
df.head(mtcars, row.names = FALSE)
df.tail(mtcars, row.names = FALSE)

# Example 3: Print first and last three rows with one max. number of decimal places
df.head(mtcars, n = 3, digits = 1)
df.head(mtcars, n = 3, digits = 1)
```

`df.long` *Converting Data Frames Between 'Wide' and 'Long' Format*

Description

The function `df.long` converts a data frame from the 'wide' data format (with repeated measurements in separate columns of the same row) to the 'long' data format (with repeated measurements in separate rows), while the function `df.wide` converts from the 'long' data format to the 'wide' data format .

Usage

```
df.long(data, ..., var = NULL, var.name = "value",
       time = c("num", "chr", "fac", "ord"), time.name = "time", idvar = "idvar",
        sort = TRUE, decreasing = FALSE, na.rm = FALSE, check = TRUE)

df.wide(data, ..., var, var.name = var, time = "time", idvar = "idvar",
        sep = "", check = TRUE)
```

Arguments

`data`	a data frame in 'wide' or 'long' format.
`...`	an expression indicating the time-invariant variable names in `data` that should be kept after converting data to the 'long' or 'wide' format. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function. Note that the `...` is not specified when all variables should be kept in the converted data frame.
`var`	a character vector (one set of variable names) or a list of character vectors (multiple sets of variables names) in the wide data format indicating the sets of time-varying variables in the wide format that correspond to single variables in the long format when using the `df.long` function. Note that all variables excluded those specified in the argument `...` are used when `var = NULL` (default), see Example 7. A character vector indicating the variable name(s) in the long format that are being split into separate variables when using the `df.wide` function.

`var.name` a character vector specifying the variable names in the long format that correspond to the sets of time-varying variables in the wide data format when using the `df.long` function or a character vector specifying the prefix of the variable names in the wide format that correspond to the time-varying variables in the long format.

`time` a character string indicating the data type of the newly created variable in the long format when using the `df.long` function, i.e., `"num"` for numeric consecutive integers starting from `0` (e.g., `0, 1, 2, 3` for a set of four variables in the wide data format), `"chr"` for a character vector, `"fac"` for a factor, and `"ord"` for a ordered factor. Note that the variable names of the set of variables in the wide data format is used when specifying `"chr"`, `"fac"`, or `"ord"` if only one set of variables is specified in the `"var"` argument. Otherwise numeric consecutive integers starting from `1` as character, factor or ordered factor are used. Or a character string indicating the variable name in the long data format that differentiates multiple records from the same group or individual when using the `df.wide` function.

`time.name` a character string indicating the name of the newly created variable in the long format when using the `df.long` function. By default, the variable is named `"time"`. Note that variable names can also be specified using the `var` when multiple sets of time-varying variables are specified in a list, e.g., `var = list(dep = c("ad", "bd"), anx = c("aa", "ba"))` (see alternative specification in Example 5).

`idvar` a character string indicating the name of the identification variable in the wide data format that is used to sort the data after converting a data frame from wide to long format when using the `df.long` function and specifying `sort = TRUE`. Note that the function will create an identification variable with consecutive integer starting from `1` if the variable specified in `idvar` is not found in `data`. Or a character string indicating the name of the identification variable in the long data format when using the `df.wide` function.

`sort` logical: if `TRUE` (default), data frame in the long format is sorted according to the identification variable specified in `idvar` when using the `df.long` function.

`decreasing` logical: if TRUE, the sort is decreasing when specifying `sort = TRUE`.

`na.rm` logical: if TRUE, rows with `NA` values for all variables in the long format that correspond to the sets of time-varying variables in the wide data format will be removed from the data when using the `df.long` function.

`check` logical: if TRUE (default), argument specification is checked.

`sep` a character string indicating a separating character in the variable names after converting data from the long format to the wide format when using the `df.wide` function. For example, the variable `value` in the long format will be split into the variables `value0`, `value1`, and `value2` when specifying `sep = ""` (default), but will be split into the variables `value_0`, `value_1`, and `value_2` when specifying `sep = "_"`.

Value

Data frame that is converted to the 'long' or 'wide' format.

Note

The function df.long uses the function melt and the function df.long uses the function dcast provided in the R package **data.table** by Tyson Barrett et al., (2025).

Author(s)

Takuya Yanagida

References

Barrett, T., Dowle, M., Srinivasan, A., Gorecki, J., Chirico, M., Hocking, T., & Schwendinger, B. (2025). data.table: Extension of 'data.frame'. R package version 1.17.8. https://CRAN.R-project.org/package=data.table

See Also

df.check, df.duplicated, df.unique, df.head, df.tail, df.merge, df.move, df.rbind, df.rename, df.sort, df.subset

Examples

```
dat.w <- data.frame(id = c(23, 55, 71),
                    gend = c("male", "female", "male"), age = c(22, 19, 26),
                    adep = c(3, 6, NA), bdep = c(5, 5, 6), cdep = c(4, NA, 5),
                    aanx = c(5, 3, 6), banx = c(NA, 7, 2), canx = c(6, NA, 8))

#————————————————————————————————————————————————————————————————————————
# Convert from 'wide' Data Format to the 'long' Data Format

# Example 1: One set of time-varying variables combined into "dep"
df.long(dat.w, var = c("adep", "bdep", "cdep"), var.name = "dep", idvar = "id")

# Example 2: Select time-invariant variables 'gend' and 'age'
df.long(dat.w, gend, age, var = c("adep", "bdep", "cdep"), var.name = "dep",
        idvar = "id")

# Example 3: Newly created variable "type" as character vector
df.long(dat.w, age, var = c("adep", "bdep", "cdep"), var.name = "dep",
        idvar = "id", time = "chr", time.name = "type")

# Example 4: User-defined variable "type"
df.long(dat.w, age, var = c("adep", "bdep", "cdep"), var.name = "dep",
        idvar = "id", time = c("pre", "post", "follow-up"), time.name = "type")

# Example 5: Two sets of time-varying variables combined into "dep" and "anx"
df.long(dat.w, age,
        var = list(c("adep", "bdep", "cdep"), c("aanx", "banx", "canx")),
        var.name = c("dep", "anx"), idvar = "id")

# Alternative specification using named lists for the argument 'var'
df.long(dat.w, age,
```

```
        var = list(dep = c("adep", "bdep", "cdep"), anx = c("aanx", "banx", "canx")),
        idvar = "id")

# Example 6: Remove rows with only NA values
df.long(dat.w, age, var = list(c("adep", "bdep", "cdep"), c("aanx", "banx", "canx")),
        idvar = "id", sort = FALSE, na.rm = TRUE)

# Example 7: Convert all variables except "age" and "gend"
df.long(dat.w, age, gend, idvar = "id")

#————————————————————————————————————————————————————————————————————————
# Convert from 'long' Data Format to the 'wide' Data Format

dat.l <- df.long(dat.w,
                 var = list(c("adep", "bdep", "cdep"), c("aanx", "banx", "canx")),
                 var.name = c("dep", "anx"), idvar = "id")

# Example 8: Time-varying variables "dep" and "anx" expanded into multiple variables
df.wide(dat.l, var = c("dep", "anx"), idvar = "id", time = "time")

# Example 9: Select time-invariant variables 'age'
df.wide(dat.l, age, var = c("dep", "anx"), idvar = "id", time = "time")

# Example 10: Variable name prefix of the  expanded variables "depre" and "anxie"
#             with separating character "."
df.wide(dat.l, var = c("dep", "anx"), var.name = c("depre", "anxie"),
        idvar = "id", time = "time", sep = ".")
```

`df.merge` *Merge Multiple Data Frames*

Description

This function merges data frames by a common column (i.e., matching variable).

Usage

```
df.merge(..., by, all = TRUE, check = TRUE, output = TRUE)
```

Arguments

`...`	a sequence of matrices or data frames and/or matrices to be merged to one.
`by`	a character string indicating the column used for merging (i.e., matching variable), see 'Details'.
`all`	logical: if `TRUE` (default), then extra rows with `NA`s will be added to the output for each row in a data frame that has no matching row in another data frame.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

There are following requirements for merging multiple data frames: First, each data frame has the same matching variable specified in the by argument. Second, matching variable in the data frames have all the same class. Third, there are no duplicated values in the matching variable in each data frame. Fourth, there are no missing values in the matching variables. Last, there are no duplicated variable names across the data frames except for the matching variable.

Note that it is possible to specify data frames matrices and/or in the argument . . . . However, the function always returns a data frame.

Value

Returns a merged data frame.

Author(s)

Takuya Yanagida

See Also

df.check, df.duplicated, df.unique, df.head, df.tail, df.long, df.wide, df.move, df.rbind, df.rename, df.sort, df.subset

Examples

```
adat <- data.frame(id = c(1, 2, 3),
                   x1 = c(7, 3, 8))

bdat <- data.frame(id = c(1, 2),
                   x2 = c(5, 1))

cdat <- data.frame(id = c(2, 3),
                   y3 = c(7, 9))

ddat <- data.frame(id = 4,
                   y4 = 6)

# Example 1: Merge 'adat', 'bdat', 'cdat', and 'ddat' by the variable 'id'
df.merge(adat, bdat, cdat, ddat, by = "id")

# Example 2: Do not show output on the console
df.merge(adat, bdat, cdat, ddat, by = "id", output = FALSE)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Error messages

adat <- data.frame(id = c(1, 2, 3),
                   x1 = c(7, 3, 8))

bdat <- data.frame(code = c(1, 2, 3),
                   x2 = c(5, 1, 3))
```

```
cdat <- data.frame(id = factor(c(1, 2, 3)),
                   x3 = c(5, 1, 3))

ddat <- data.frame(id = c(1, 2, 2),
                   x2 = c(5, 1, 3))

edat <- data.frame(id = c(1, NA, 3),
                   x2 = c(5, 1, 3))

fdat <- data.frame(id = c(1, 2, 3),
                   x1 = c(5, 1, 3))

# Error 1: Data frames do not have the same matching variable specified in 'by'.
df.merge(adat, bdat, by = "id")

# Error 2: Matching variable in the data frames do not all have the same class.
df.merge(adat, cdat, by = "id")

# Error 3: There are duplicated values in the matching variable specified in 'by'.
df.merge(adat, ddat, by = "id")

# Error 4: There are missing values in the matching variable specified in 'by'.
df.merge(adat, edat, by = "id")

# Error 5: There are duplicated variable names across data frames.
df.merge(adat, fdat, by = "id")

## End(Not run)
```

`df.move` *Move Variable(s) in a Data Frame*

Description

This function moves variables to a different position in the data frame, i.e., changes the column positions in the data frame. By default, variables specified in the first argument `...` are moved to the first position in the data frame specified in the argument `data`.

Usage

```
df.move(data, ..., before = NULL, after = NULL, first = TRUE, check = TRUE)
```

Arguments

`data`	a data frame.
`...`	an expression indicating the variable names in `data` to move. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see Details in the `df.subset` function.

before	a character string indicating a variable in data. Variable(s) specified in ... are moved to the left-hand side of this variable.
after	a character string indicating a variable in data. Variable(s) specified in ... are moved to the right-hand side of this variable.
first	logical: if TRUE (default), variable(s) specified in ... will be moved to the first position in 'data', if FALSE, variable(s) specified in ... will be moved to the last position in 'data'.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns the data frame in data with columns in a different place.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

df.check, df.duplicated, df.unique, df.head, df.tail, df.long, df.wide, df.merge, df.rbind, df.rename, df.sort, df.subset

Examples

```
# Example 1: Move variables 'hp' and 'am' to the first position
df.move(mtcars, hp, am)

# Example 2: Move variables 'hp' and 'am' to the last position
df.move(mtcars, hp, am, first = FALSE)

# Example 3: Move variables 'hp' and 'am' to the left-hand side of 'disp'
df.move(mtcars, hp, am, before = "disp")

# Example 4: Move variables 'hp' and 'am' to the right-hand side of 'disp'
df.move(mtcars, hp, am, after = "disp")
```

df.rbind *Combine Data Frames by Rows, Filling in Missing Columns*

Description

This function takes a sequence of data frames and combines them by rows, while filling in missing columns with NAs.

Usage

```
df.rbind(...)
```

Arguments

...	a sequence of data frame to be row bind together. This argument can be a list of data frames, in which case all other arguments are ignored. Any NULL inputs are silently dropped. If all inputs are NULL, the output is also NULL.

Details

This is an enhancement to rbind that adds in columns that are not present in all inputs, accepts a sequence of data frames, and operates substantially faster.

Column names and types in the output will appear in the order in which they were encountered.

Unordered factor columns will have their levels unified and character data bound with factors will be converted to character. POSIXct data will be converted to be in the same time zone. Array and matrix columns must have identical dimensions after the row count. Aside from these there are no general checks that each column is of consistent data type.

Value

Returns a single data frame

Note

This function is a copy of the rbind.fill() function in the **plyr** package by Hadley Wickham.

Author(s)

Takuya Yanagida

References

Wickham, H. (2011). The split-apply-combine strategy for data analysis. *Journal of Statistical Software, 40*, 1-29. https://doi.org/10.18637/jss.v040.i01

Wickham, H. (2019). plyr: Tools for Splitting, Applying and Combining Data. R package version 1.8.5.

See Also

df.check, df.duplicated, df.unique, df.head, df.tail, df.long, df.wide, df.merge, df.move, df.rename, df.sort, df.subset

Examples

```
adat <- data.frame(id = c(1, 2, 3), a = c(7, 3, 8), b = c(4, 2, 7))

bdat <- data.frame(id = c(4, 5, 6), a = c(2, 4, 6), c = c(4, 2, 7))

cdat <- data.frame(id = c(7, 8, 9), a = c(1, 4, 6), d = c(9, 5, 4))

# Example 1
df.rbind(adat, bdat, cdat)
```

---

df.rename *Rename Columns in a Matrix or Variables in a Data Frame*

Description

This function renames columns in a matrix or variables in a data frame by (1) using old_name = new_name, by using the functions toupper, tolower, sub, and gsub, or (3) by specifying a character vector indicating the column(s) or variable(s) to be renamed (argument from) and a character vector indicating the corresponding replacement values (argument to).

Usage

```
df.rename(data, ..., from, to, check = TRUE)
```

Arguments

data	a matrix or data frame.
...	old_name = new_name when from = NULL and to = NULL, or one of the functions toupper, tolower, sub, and gsub. Note that a tilde (~) needs to be specified before when using a function, e.g., ~toupper or ~gsub("_", ".") .
from	a character vector indicating the column(s) or variable(s) to be renamed.
to	a character vector indicating the corresponding replacement values for the column(s) or variable(s) specified in the argument name.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns the matrix or data frame data with renamed columns or variables.

Author(s)

Takuya Yanagida

See Also

df.check, df.duplicated, df.unique, df.head, df.tail, df.long, df.wide, df.merge, df.move, df.rbind, df.sort, df.subset

Examples

```
#-----------------------------------------------------------------------
# Rename using variable names

# Example 1a: Rename 'cyl' in 'mtcars' to 'cylinder' using 'old_name = new_name'
df.rename(mtcars, cyl = cylinder)

# Example 1b: Rename 'cyl' in 'mtcars' to 'cylinder' using 'from' and 'to'
df.rename(mtcars, from = "cyl", to = "cylinder")

# Example 2a: Rename 'cyl' and 'wt' in 'mtcars' to 'cylinder' and 'weight'
# using 'old_name = new_name'
df.rename(mtcars, cyl = cylinder, wt = weight)

# Example 2b: Rename 'cyl' and 'wt' in 'mtcars' to 'cylinder' and 'weight'
# using using 'from' and 'to'
df.rename(mtcars, from = c("cyl", "wt"), to = c("cylinder", "weight"))

#-----------------------------------------------------------------------
# Rename using functions

# Example 3: Convert all variable names to lowercase
df.rename(iris, ~tolower)

# Example 4: Replace all '.' with '_'
# Note, the argument fixed is set to TRUE by default.
df.rename(iris, ~gsub(".", "_"))

# Example 5: Replace all 'S' with 'P'
df.rename(iris, ~gsub("S", "P"))

# Example 6: Replace all 'S' with 'P', ignore case during matching
df.rename(iris, ~gsub("S", "P", ignore.case = TRUE))
```

---

df.sort *Data Frame Sorting*

Description

This function arranges a data frame in increasing or decreasing order according to one or more variables.

Usage

```
df.sort(data, ..., decreasing = FALSE, check = TRUE)
```

Arguments

`data`	a data frame.
`...`	a sorting variable or a sequence of sorting variables which are specified without quotes `''` or double quotes `""`.
`decreasing`	logical: if `TRUE`, the sort is decreasing.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns data frame `data` sorted according to the variables specified in `...`, a matrix will be coerced to a data frame.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Knuth, D. E. (1998) *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.

See Also

`df.check`, `df.duplicated`, `df.unique`, `df.head`, `df.tail`, `df.long`, `df.wide`, `df.merge`, `df.move`, `df.rbind`, `df.rename`, `df.subset`

Examples

```
# Example 1: Sort data frame 'mtcars' by 'mpg' in increasing order
df.sort(mtcars, mpg)

# Example 2: Sort data frame 'mtcars' by 'mpg' in decreasing order
df.sort(mtcars, mpg, decreasing = TRUE)

# Example 3: Sort data frame 'mtcars' by 'mpg' and 'cyl' in increasing order
df.sort(mtcars, mpg, cyl)

# Example 4: Sort data frame 'mtcars' by 'mpg' and 'cyl' in decreasing order
df.sort(mtcars, mpg, cyl, decreasing = TRUE)
```

`df.subset` *Subsetting Data Frames*

Description

This function returns subsets of data frames which meet conditions.

Usage

```
df.subset(data, ..., subset = NULL, drop = TRUE, check = TRUE)
```

Arguments

`data`	a data frame.
`...`	an expression indicating variables to select from the data frame specified in `data`. See Details for the list of operators used in this function, i.e., `+`, `-`, `~`, `:`, `::`, and `!`. Note that all variables are selected if the argument `...` is not specified.
`subset`	a logical expression indicating rows to keep, e.g., `var == 1`, `var1 == 1 & var2 == 3`, or `gender == "female"`. By default, all rows of the data frame specified in `data` are kept. Note that logical queries for rows resulting in missing values are not select.
`drop`	logical: if `TRUE` (default), data frame with a single column is converted into a vector.
`check`	logical: if `TRUE` (default), argument specification is checked.

Details

The argument `...` is used to specify an expression indicating the variables to select and/or remove from the data frame specified in `data`. There are six operators which can be used in the expression `...`:

Plus (+) Operator The plus operator is used to select variables matching a prefix from the data frame specified in `data`. For example, `df.subset(dat, +x)` selects all variables with the prefix `x`. Note that this operator is equivalent to the function `starts_with()` from the **tidyselect** package.

Minus (-) Operator The minus operator is used to select variables matching a suffix from the data frame specified in `data`. For example, `df.subset(dat, -y)` selects all variables with the suffix `y`. Note that this operator is equivalent to the function `ends_with()` from the **tidyselect** package.

Tilde (~) Operator The tilde operator is used to select variables containing a word from the data frame specified in `data`. For example, `df.subset(dat, ~al)` selects all variables with the word `al`. Note that this operator is equivalent to the function `contains()` from the **tidyselect** package.

Colon (`:`) operator The colon operator is used to select a range of consecutive variables from the data frame specified in `data`. For example, `df.subset(dat, x:z)` selects all variables from `x` to `z`. Note that this operator is equivalent to the `:` operator from the `select` function in the **dplyr** package.

Double Colon (`::`) Operator The double colon operator is used to select numbered variables from the data frame specified in `data`. For example, `df.subset(dat, x1::x3)` selects the variables `x1`, `x2`, and `x3`. Note that this operator is similar to the function `num_range()` from the **tidyselect** package.

Exclamation Point (`!`) Operator The exclamation point operator is used to drop variables from the data frame specified in the argument `data` or for taking the complement of a set of variables. For example, `df.subset(dat, !x)` selects all variables except the variable `x`, `df.subset(dat, !~x)` selects all variables except variables with the prefix `x`, or `df.subset(dat, x1:x10, !x3:x5)` selects all variables from `x1` to `x10` but excludes all variables from `x3` to `x5`. Note that this operator is equivalent to the `!` operator from the `select` function in the **dplyr** package.

Operators can be combined within the same function call. For example, `df.subset(dat, +x, -y, !x2:x4, z)` selects all variables with the prefix `x` and with the suffix `y` but excludes variables from `x2` to `x4` and selects variable `z`.

Value

Returns a data frame containing the variables and rows selected in the argument `...` and rows selected in the argument `subset`.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`df.check`, `df.duplicated`, `df.unique`, `df.head`, `df.tail`, `df.long`, `df.wide`, `df.merge`, `df.move`, `df.rbind`, `df.rename`, `df.sort`,

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————————
# Select Single Variables

# Example 1: Select 'Sepal.Length' and 'Petal.Width'
df.subset(iris, Sepal.Length, Petal.Width)

#————————————————————————————————————————————————————————————————————————
# Select Rows
```

```
# Example 2a: Select all variables, select rows with 'Species' equal 'setosa'
df.subset(iris, subset = Species == "setosa")

# Example 2b: Select all variables, select rows with 'Petal.Length' smaller 1.2
df.subset(iris, subset = Petal.Length < 1.2)

# Example 2C: Select 'Sepal.Length' and 'Petal.Width',
# select rows with 'Species' equal 'setosa'
df.subset(iris, Sepal.Length, Petal.Width, subset = Species == "setosa")

#————————————————————————————————————————————————————————————————————————
# Select Variables Matching a Prefix using the + Operator

# Example 3: Select variables with prefix 'Petal'
df.subset(iris, +Petal)

#————————————————————————————————————————————————————————————————————————
# Select Variables Matching a Suffix using the - Operator

# Example 4: Select variables with suffix 'Width'
df.subset(iris, -Width)

#————————————————————————————————————————————————————————————————————————
# Select Variables Containing a Word using the ~ Operator

# Example 5: Select variables containing 'al'
df.subset(iris, ~al)

#————————————————————————————————————————————————————————————————————————
# Select Consecutive Variables using the : Operator

# Example 6: Select all variables from 'Sepal.Width' to 'Petal.Width'
df.subset(iris, Sepal.Width:Petal.Width)

#————————————————————————————————————————————————————————————————————————
# Select Numbered Variables using the :: Operator

# Example 7: Select all variables from 'x1' to 'x3' and 'y1' to 'y3'
df.subset(anscombe, x1::x3, y1::y3)

#————————————————————————————————————————————————————————————————————————
# Drop Variables using the ! Operator

# Example 8a: Select all variables except 'Sepal.Width'
df.subset(iris, !Sepal.Width)

# Example 8b: Select all variables except variables with prefix 'Petal'
df.subset(iris, !+Petal)

# Example 8c: Select all variables except variables with suffix 'Width'
df.subset(iris, !-Width)
```

```
# Example 8d: Select all variables except 'Sepal.Width' to 'Petal.Width'
df.subset(iris, !Sepal.Width:Petal.Width)

#————————————————————————————————————————————————————————————————————
# Combine +, -, !, and : Operators

# Example 9: Select variables with prefix 'x' and suffix '3', but exclude
# variables from 'x2' to 'x3'
df.subset(anscombe, +x, -3, !x2:x3)

## End(Not run)
```

`difftest.chibarsq` *Chi-Bar-Square Difference Test*

Description

This function performs the chi-bar-square difference test to compare the random intercept cross-lagged panel model (RI-CLPM) and traditional cross-lagged panel model (CLPM) as discussed in Hamaker et al. (2015).

Usage

```
difftest.chibarsq(clpm, riclpm, alpha = 0.05, digits = 2, p.digits = 3,
                  write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`clpm`	an object of class lavaan, i.e., a fitted random intercept cross-lagged panel model (RI-CLPM) with the variance and covariances of latent intercept factors fixed to zero. Note that a RI-CLPM with the variance of all random intercepts fixed to zero is statistically equivalent to the traditional cross-lagged panel model (CLPM).
`riclpm`	an object of class lavaan, i.e., a fitted random intercept cross-lagged panel model with variance and covariances of latent intercept factors freely estimated.
`alpha`	a numeric value indicating the type-I-risk, α.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-values.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Random Intercept Cross-Lagged Panel Model (RI-CLPM) The RI-CLPM is an extension of the traditional cross-lagged panel model that disentangles the within-person process from stable between-person differences (Hamaker et al., 2015). In a bivariate RI-CLPM, each variable x and y is decomposed into a stable time-invariant trait-like component, captured with random intercept factors denoted by κ for variable x and ω for variable y (see Figure 1 in Hamaker et al., 2015). Note that the CLPM is nested under the RI-CLPM, i.e., the RI-CLPM is statistically equivalent to the CLPM when fixing the variance of all random intercepts and their covariances to zero.

Chi-Bar-Square Difference Test The $\bar{\chi}^2$ difference test is used to compare the fit of the nested models CLPM and RI-CLPM based on a mixture of chi-square distributions to test the null hypothesis e.g., $H_0 : Var_\kappa = 0, Var_\omega = 0, Cov_{\kappa,\omega} = 0$. The chi-bar-square distribution is a weighted sum of different chi-square distributions with varying degrees of freedom resulting from parameters fixed at the boundaries of the parameter space as variances are non-negative values (Stoel et al., 2016).

Chi-Square Difference Test The regular χ^2 difference test is conservative due to ignoring the mixture distribution, i.e., if it is statistically significant, we are certain that the chi-bar-square difference test will be significant too, while the reverse will not be the case. Accordingly, if researchers find it more important to detect a true CLPM than a true RI-CLPM, it is advised to use the regular chi-square difference test (Sukpan & Kuiper, 2026). It should also be mentioned that estimating a RI-CLPM when the CLPM is the true model may reduce statistical power due to estimating additional parameters, but it does not introduce bias (see Table 4 in Scott, 2021), while estimating a CLPM when the RI-CLPM is the true model introduces bias (see Table 3 in Scott, 2021).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	data frame including all variables used in the analysis, i.e., indicators for the factor, grouping variable and cluster variable
`args`	specification of function arguments
`model.fit`	list of fitted lavaan objects specified in the argument `riclpm` and `clpm`
`result`	list with result tables, i.e., `difftest` for the chi-bar-square difference test and `weights` for the weights for the mixture of chi-square mixture distribution

Note

This function is based on modified copies of the function `ChiBarSq.DiffTest` from the **ChiBarSq.DiffTest** package by Rebecca M. Kuiper.

Author(s)

Takuya Yanagida

References

Hamaker, E. L., Kuiper, R. M., & Grasman, R. P. (2015). A critique of the cross-lagged panel model. *Psychological Methods, 20*(1), 102-116. https://doi.org/10.1037/a0038889

Kuiper R (2026). *ChiBarSq.DiffTest: Chi-bar-square difference test of the RI-CLPM versus the CLPM and more general*. R package version 0.0.0.9000. https://github.com/rebeccakuiper/ChiBarSq.DiffTest

Mulder, J. D., & Hamaker, E. L. (2021). Three extensions of the random intercept cross-lagged panel model. *Structural Equation Modeling: A Multidisciplinary Journal, 28*(4), 638-648. https://doi.org/10.1080/10705511.2020.1784738

Scott, P. W. (2021). Accounting for time-varying inter-individual differences in trajectories when assessing cross-lagged models. *Structural Equation Modeling, 28*(3), 365-375. https://doi.org/10.1080/10705511.2020.1819815

Stoel, R. D., Garre, F. G., Dolan, C., & van den Wittenboer, G. (2006). On the likelihood ratio test in structural equation modeling when parameters are subject to boundary constraints. *Psychological Methods, 11*(4), 439-455. https://doi.org/10.1037/1082-989X.11.4.439

Sukpan, C., & Kuiper, R. M. (2026). Selecting the correct RI-CLPM using chi-square-type tests and AIC-type criteria. *Structural Equation Modeling: A Multidisciplinary Journal*, 1-14. https://doi.org/10.1080/10705511.2025.2592831

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————————
# Step-wise Procedure (Sukpan & Kuiper, 2026)
#
# Note that only the first step is shown in this example:
# - CLPM versus RI-CLPM(Kappa)
# - CLPM versus RI-CLPM(Omega)
#
# Model specification based on code provided on the accompanying website of
# Mulder and Hamaker (2021)

#————————————————————————————————————
## Model Specification: Cross-Lagged Panel Model (CLPM)
## i.e., Var(Kappa) = 0, Var(Omega) = 0, Cov(Kappa, Omega) = 0

mod.clpm <- '
  # Create between components (random intercepts)
  RIx =~ 1*x1 + 1*x2 + 1*x3
  RIy =~ 1*y1 + 1*y2 + 1*y3

  # Create within-person centered variables
  wx1 =~ 1*x1
  wx2 =~ 1*x2
  wx3 =~ 1*x3
  wy1 =~ 1*y1
  wy2 =~ 1*y2
  wy3 =~ 1*y3

  # Estimate lagged effects between within-person centered variables
```

```
  wx2 + wy2 ~ wx1 + wy1
  wx3 + wy3 ~ wx2 + wy2

  # Estimate covariance between within-person centered variables at first wave
  wx1 ~~ wy1 # Covariance

  # Estimate covariances between residuals of within-person centered variables
  wx2 ~~ wy2
  wx3 ~~ wy3

  # Fix variance and covariance of random intercepts to zero, i.e.,
  RIx ~~ 0*RIx
  RIy ~~ 0*RIy
  RIx ~~ 0*RIy

  # Estimate (residual) variance of within-person centered variables
  wx1 ~~ wx1
  wy1 ~~ wy1
  wx2 ~~ wx2
  wy2 ~~ wy2
  wx3 ~~ wx3
  wy3 ~~ wy3
'

#————————————————————————————————————
## Model Specification: Random Intercept Cross-Lagged Panel Model RI-CLPM(Kappa)
## i.e., Var(Kappa) > 0, Var(Omega) = 0, Cov(Kappa, Omega) = 0

mod.ri.clpm.k <- '
  # Create between components (random intercepts)
  RIx =~ 1*x1 + 1*x2 + 1*x3
  RIy =~ 1*y1 + 1*y2 + 1*y3

  # Create within-person centered variables
  wx1 =~ 1*x1
  wx2 =~ 1*x2
  wx3 =~ 1*x3
  wy1 =~ 1*y1
  wy2 =~ 1*y2
  wy3 =~ 1*y3

  # Estimate lagged effects between within-person centered variables
  wx2 + wy2 ~ wx1 + wy1
  wx3 + wy3 ~ wx2 + wy2

  # Estimate covariance between within-person centered variables at first wave
  wx1 ~~ wy1

  # Estimate covariances between residuals of within-person centered variables
  wx2 ~~ wy2
  wx3 ~~ wy3

  # Fix variance of random intercept RIy and covariance with RIx to zero
```

```
  RIx ~~ RIx
  RIy ~~ 0*RIy
  RIx ~~ 0*RIy

  # Estimate (residual) variance of within-person centered variables
  wx1 ~~ wx1
  wy1 ~~ wy1
  wx2 ~~ wx2
  wy2 ~~ wy2
  wx3 ~~ wx3
  wy3 ~~ wy3
'

#————————————————————————————————————
## Model Specification: Random Intercept Cross-Lagged Panel Model RI-CLPM(Omega)
## i.e., Var(Kappa) = 0, Var(Omega) > 0, Cov(Kappa, Omega) = 0

mod.ri.clpm.o <- '
  # Create between components (random intercepts)
  RIx =~ 1*x1 + 1*x2 + 1*x3
  RIy =~ 1*y1 + 1*y2 + 1*y3

  # Create within-person centered variables
  wx1 =~ 1*x1
  wx2 =~ 1*x2
  wx3 =~ 1*x3
  wy1 =~ 1*y1
  wy2 =~ 1*y2
  wy3 =~ 1*y3

  # Estimate lagged effects between within-person centered variables
  wx2 + wy2 ~ wx1 + wy1
  wx3 + wy3 ~ wx2 + wy2

  # Estimate covariance between within-person centered variables at first wave
  wx1 ~~ wy1 # Covariance

  # Estimate covariances between residuals of within-person centered variables
  wx2 ~~ wy2
  wx3 ~~ wy3

  # Fix variance of random intercept RIx and covariance with RIy to zero
  RIx ~~ 0*RIx
  RIy ~~ RIy
  RIx ~~ 0*RIy

  # Estimate (residual) variance of within-person centered variables
  wx1 ~~ wx1
  wy1 ~~ wy1
  wx2 ~~ wx2
  wy2 ~~ wy2
  wx3 ~~ wx3
  wy3 ~~ wy3
```

```
'

#═══════════════════════════════════
## Estimate Models
#
# Note that the example analysis cannot be conduct as the data set 'data'
# is not available.

# CLPM
fit.clpm <- lavaan(mod.clpm, data = data, estimator = "MLR")

# RI-CLPM(Kappa)
fit.ri.clpm.k <- lavaan(mod.ri.clpm.k, data = data, estimator = "MLR")

# RI-CLPM(Omega)
fit.ri.clpm.o <- lavaan(mod.ri.clpm.o, data = data, estimator = "MLR")

#···................
# Chi-Bar-Square Difference Test

# CLPM vs. RI-CLPM(Kappa)
difftest.chibarsq(fit.clpm, fit.ri.clpm.k)

# CLPM vs. RI-CLPM(Omega)
difftest.chibarsq(fit.clpm, fit.ri.clpm.o)

## End(Not run)
```

`dominance` *Dominance Analysis*

Description

This function conducts dominance analysis (Budescu, 1993; Azen & Budescu, 2003) for linear models estimated by using the `lm()` function to determine the relative importance of predictor variables. By default, the function reports general dominance, but conditional and complete dominance can be requested by specifying the argument `print`.

Usage

```
dominance(model, print = c("all", "gen", "cond", "comp"), digits = 3,
          write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted model of class `lm`.
`print`	a character vector indicating which results to show on the console, i.e. `"all"` for all results, `"gen"` for general dominance, `"cond"` for conditional dominance, and `"comp"` for complete dominance.

`digits`	an integer value indicating the number of decimal places to be used for displaying results. Note that the percentage relative importance of predictors are printed with `digits` minus 1 decimal places.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Details

Dominance analysis (Budescu, 1993; Azen & Budescu, 2003) is used to determine the relative importance of predictor variables in a statistical model by examining the additional contribution of predictors in *R*-squared relative to each other in all of the possible $2^{(p-2)}$ subset models with p being the number of predictors. Three levels of dominance can be established through pairwise comparison of all predictors in a regression model:

Complete Dominance A predictor completely dominates another predictor if its additional contribution in *R*-Squared is higher than that of the other predictor across all possible subset models that do not include both predictors. For example, in a regression model with four predictors, X_1 completely dominates X_2 if the additional contribution in *R*-squared for X_1 is higher compared to X_2 in (1) the null model without any predictors, (2) the model including X_3, (3) the model including X_4, and (4) the model including both X_3 and X_4. Note that complete dominance cannot be established if one predictor's additional contribution is greater than the other's for some, but not all of the subset models. In this case, dominance is undetermined and the result will be `NA`

Conditional Dominance A predictor conditionally dominates another predictor if its average additional contribution in *R*-squared is higher within each model size than that of the other predictor. For example, in a regression model with four predictors, X_1 conditionally dominates X_2 if the average additional contribution in *R*-squared is higher compared to X_2 in (1) the null model without any predictors, (2) the four models including one predictor, (3) the six models including two predictors, and (4) the four models including three predictors.

General Dominance A predictor generally dominates another predictor if its overall averaged additional contribution in *R*-squared is higher than that of the other predictor. For example, in a regression model with four predictors, X_1 generally dominates X_2 if the average across the four conditional values (i.e., null model, model with one predictor, model with two predictors, and model with three predictors) is higher than that of X_2. Note that the general dominance measures represent the proportional contribution that each predictor makes to the *R*-squared since their sum across all predictors equals the *R*-squared of the full model.

The three levels of dominance are related to each other in a hierarchical fashion: Complete dominance implies conditional dominance, which in turn implies general dominance. However, the converse may not hold for more than three predictors. That is, general dominance does not imply conditional dominance, and conditional dominance does not necessarily imply complete dominance.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`args`	specification of function arguments
`result`	list with results, i.e., `gen` for general dominance, `cond` for conditional dominance, `comp` for complete dominance, and `condtsat` for the statistics of the conditional dominance

Note

This function is based on the `domir` function from the `domir` package (Luchman, 2023).

Author(s)

Takuya Yanagida

References

Azen, R., & Budescu, D. V. (2003). The dominance analysis approach for comparing predictors in multiple regression. *Psychological Methods, 8*(2), 129–148. https://doi.org/10.1037/1082-989X.8.2.129

Budescu, D. V. (1993). Dominance analysis: A new approach to the problem of relative importance of predictors in multiple regression. *Psychological Bulletin, 114*(3), 542–551. https://doi.org/10.1037/0033-2909.114.3.542

Luchman J (2023). *domir: Tools to support relative importance analysis*. R package version 1.0.1, https://CRAN.R-project.org/package=domir.

See Also

`dominance.manual`, `coeff.std`, `write.result`

Examples

```
#————————————————————————————————————————————————————————————————————
# Example 1: Dominance Analysis for a Linear Model

# Example 1
mod <- lm(mpg ~ cyl + disp + hp, data = mtcars)
dominance(mod)

# Print all results
dominance(mod, print = "all")

## Not run:
#————————————————————————————————————————————————————————————————————
# Write results into a Text or Excel File
```

```
# Example 2a: Text file
dominance(mod, write = "Dominance.txt", output = FALSE)

# Example 2b: Excel file
dominance(mod, write = "Dominance.xlsx", output = FALSE)
## End(Not run)
```

`dominance.manual` *Dominance Analysis, Manually Inputting a Correlation Matrix*

Description

This function conducts dominance analysis (Budescu, 1993; Azen & Budescu, 2003) based on a (model-implied) correlation matrix of the manifest or latent variables. Note that the function only provides general dominance.

Usage

```
dominance.manual(x, out = NULL, digits = 3, write = NULL, append = TRUE,
                 check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with the (model-implied) correlation matrix of the manifest or latent variables. Note that column names need to represent the variables names in `x`.
`out`	a character string representing the outcome variable. By default, the first row and column represents the outcome variable.
`digits`	an integer value indicating the number of decimal places to be used for displaying results. Note that the percentage relative importance of predictors are printed with `digits` minus 1 decimal places.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	correlation matrix specified in `x`
`args`	specification of function arguments
`result`	results table for the general dominance

Note

This function implements the function provided in Appendix 1 of Gu (2022) and copied the function `combinations()` from the `gtools` package (Bolker, Warnes, & Lumley, 2022).

Author(s)

Takuya Yanagida

References

Azen, R., & Budescu, D. V. (2003). The dominance analysis approach for comparing predictors in multiple regression. *Psychological Methods, 8*(2), 129–148. https://doi.org/10.1037/1082-989X.8.2.129

Bolker, B., Warnes, G., & Lumley, T. (2022). *gtools: Various R Programming Tools*. R package version 3.9.4, https://CRAN.R-project.org/package=gtools

Budescu, D. V. (1993). Dominance analysis: A new approach to the problem of relative importance of predictors in multiple regression. *Psychological Bulletin, 114*(3), 542–551. https://doi.org/10.1037/0033-2909.114.3.542

Gu, X. (2022). Assessing the relative importance of predictors in latent regression models. *Structural Equation Modeling: A Multidisciplinary Journal, 4*, 569-583. https://doi.org/10.1080/10705511.2021.2025377

See Also

`dominance`, `coeff.std`, `write.result`

Examples

```
#————————————————————————————————————————————————————————————————————
# Linear model

# Example 1a: Dominance analysis, 'mpg' predicted by 'cyl', 'disp', and 'hp'
dominance.manual(cor(mtcars[, c("mpg", "cyl", "disp", "hp")]))

# Example 1b: Equivalent results using the dominance() function
dominance(lm(mpg ~ cyl + disp + hp, data = mtcars))

# Example 1c: Dominance analysis, 'hp' predicted by 'mpg', 'cyl', and 'disp'
```

```
dominance.manual(cor(mtcars[, c("mpg", "cyl", "disp", "hp")]), out = "hp")

## Not run:
# Example 1d: Write results into a text file
dominance.manual(cor(mtcars[, c("mpg", "cyl", "disp", "hp")]),
                 write = "Dominance_Manual.txt")
## End(Not run)

#————————————————————————————————————————————————————————————————————————
# Structural equation modeling

library(lavaan)

#————————————————————————————————————
# Example 2: Latent variables

# Model specification
model <- '# Measurement model
          ind60 =~ x1 + x2 + x3
          dem60 =~ y1 + y2 + y3 + y4
          dem65 =~ y5 + y6 + y7 + y8
          # regressions
          ind60 ~ dem60 + dem65'

# Model estimation
fit <- sem(model, data = PoliticalDemocracy)

# Model-implied correlation matrix of the latent variables
fit.cor <- lavInspect(fit, what = "cor.lv")

# Dominance analysis
dominance.manual(fit.cor)

#————————————————————————————————————
# Example 3: Latent and manifest variables

# Model specification, convert manifest to latent variable
model <- '# Measurement model
          ind60 =~ x1 + x2 + x3
          dem60 =~ y1 + y2 + y3 + y4
          # Manifest as latent variable
          ly5 =~ 1*y5
          y5 ~~ 0*y5
          # Regressions
          ind60 ~ dem60 + ly5'

# Model estimation
fit <- sem(model, data = PoliticalDemocracy)

# Model-implied correlation matrix of the latent variables
fit.cor <- lavInspect(fit, what = "cor.lv")

# Dominance analysis
```

```
dominance.manual(fit.cor)

#————————————————————————————————————————————————————————————————————————
# Example 4: Multilevel modeling

# Model specification
model <- 'level: 1
            fw =~ y1 + y2 + y3
            # Manifest as latent variables
            lx1 =~ 1*x1
            lx2 =~ 1*x2
            lx3 =~ 1*x3
            x1 ~~ 0*x1
            x2 ~~ 0*x2
            x3 ~~ 0*x3
            # Regression
            fw ~ lx1 + lx2 + lx3
          level: 2
            fb =~ y1 + y2 + y3
            # Manifest as latent variables
            lw1 =~ 1*w1
            lw2 =~ 1*w2
            # Regression
            fb ~ lw1 + lw2'

# Model estimation
fit <- sem(model, data = Demo.twolevel, cluster = "cluster")

# Model-implied correlation matrix of the latent variables
fit.cor <- lavInspect(fit, what = "cor.lv")

# Dominance analysis Within
dominance.manual(fit.cor$within)

# Dominance analysis Between
dominance.manual(fit.cor$cluster)

## Not run:
#————————————————————————————————————————————————————————————————————————
# Example 5: Mplus
#
# In Mplus, the model-implied correlation matrix of the latent variables
# can be requested by OUTPUT: TECH4 and imported into R by using the
# MplusAuomtation package, for example:

library(MplusAutomation)

# Read Mplus output
output <- readModels()

# Extract model-implied correlation matrix of the latent variables
fit.cor <- output$tech4$latCorEst
```

```
## End(Not run)
```

---

effsize *Effect Sizes for Categorical Variables*

Description

This function computes effect sizes for one or more than one categorical variable, i.e., (adjusted) phi coefficient, (bias-corrected) Cramer's *V*, (bias-corrected) Tschuprow's *T*, (adjusted) Pearson's contingency coefficient, Cohen's *w*), and *Fei*. By default, the function computes *Fei* based on a chi-square goodness-of-fit test for one categorical variable, phi coefficient based on a chi-square test of independence for two dichotomous variables, and Cramer's *V* based on a chi-square test of independence for two variables with at least one polytomous variable.

Usage

```
effsize(data, ..., type = c("phi", "cramer", "tschuprow", "cont", "w", "fei"),
        alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
        adjust = TRUE, indep = TRUE, p = NULL, digits = 3, as.na = NULL,
        write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a vector, factor or data frame.
`...`	an expression indicating the variable names in `data`, e.g., `effsize(dat, x1, x2)`. When specifying more than one variable, the first variable is always the focal variable in the Chi-square test of independence which association with all other variables is investigated. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`type`	a character string indicating the type of effect size, i.e., `phi` for phi coefficient, `cramer` for Cramer's V, `tschuprow` for Tschuprow's T, `cont` for Pearson's contingency coefficient, `w` for Cohen's w, and `Fei` for Fei.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`adjust`	logical: if `TRUE` (default), phi coefficient and Pearson's contingency coefficient are adjusted by relating the coefficient to the possible maximum, or Cramer's *V* and Tschuprow's *T* are corrected for small-sample bias.
`indep`	logical: if `TRUE`, effect size computation is based on a chi-square test of independence (default when specifying two variable, if `FALSE` effect size computation is based on a chi-square goodness-of-fit test (default when specifying one variable).
`p`	a numeric vector specifying the expected proportions in each category of the categorical variable when conducting a chi-square goodness-of-fit test. By default, the expected proportions in each category are assumed to be equal.

digits	an integer value indicating the number of decimal places digits to be used for displaying the results.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame with variables used in the current analysis
args	specification of function arguments
result	result table

Note

This function is based on modified copies of the functions chisq_to_phi, chisq_to_cramers_v, chisq_to_tschuprows_t, chisq_to_pearsons_c, chisq_to_cohens_w, and chisq_to_fei from the **effectsize** package (Ben-Shachar, Lüdecke & Makowski, 2020).

Author(s)

Takuya Yanagida

References

Bergsma, W. (2013). A bias correction for Cramer's V and Tschuprow's T. *Journal of the Korean Statistical Society, 42*, 323-328. https://doi.org/10.1016/j.jkss.2012.10.002

Ben-Shachar M. S., Lüdecke D., Makowski D. (2020). effectsize: Estimation of Effect Size Indices and Standardized Parameters. *Journal of Open Source Software, 5* (56), 2815. https://doi.org/10.21105/joss.02815

Ben-Shachar, M. S., Patil, I., Theriault, R., Wiernik, B. M., Lüdecke, D. (2023). Phi, Fei, Fo, Fum: Effect sizes for categorical data that use the chi-squared statistic. *Mathematics, 11*, 1982. https://doi.org/10.3390/math11091982

Cureton, E. E. (1959). Note on Phi/Phi max. *Psychometrika, 24*, 89-91.

Davenport, E. C., & El-Sanhurry, N. A. (1991). Phi/Phimax: Review and synthesis. *Educational and Psychological Measurement, 51*, 821-828. https://doi.org/10.1177/001316449105100403

Sakoda, J.M. (1977). Measures of association for multivariate contingency tables. *Proceedings of the Social Statistics Section of the American Statistical Association (Part III)*, 777-780.

See Also

cor.matrix, cohens.d

Examples

```
# Example 1a: Phi coefficient for 'vs' and 'am'
effsize(mtcars, vs, am)

# Example 1b: Alternative specification without using the '...' argument
effsize(mtcars[, c("vs", "am")])

# Example 2: Bias-corrected Cramer's V for 'gear' and 'carb'
effsize(mtcars, gear, carb)

# Example 3: Cramer's V (without bias-correction) for 'gear' and 'carb'
effsize(mtcars, gear, carb, adjust = FALSE)

# Example 4: Adjusted Pearson's contingency coefficient for 'gear' and 'carb'
effsize(mtcars, gear, carb, type = "cont")

# Example 5: Fei for 'gear'
effsize(mtcars, gear)

# Example 6a: Bias-corrected Cramer's V for 'cyl' and 'vs', 'am', 'gear', and 'carb'
effsize(mtcars, cyl, vs:carb)

# Example 6b:  Alternative specification without using the '...' argument
effsize(mtcars[, c("cyl", "vs", "am", "gear", "carb")])

## Not run:
#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 7a: Write Results into a text file
effsize(mtcars, cyl, vs:carb, write = "Cramer.txt")

# Example 7b: Write Results into an Excel file
effsize(mtcars, cyl, vs:carb, write = "Cramer.xlsx")

## End(Not run)
```

---

freq *Frequency Table*

Description

This function computes a frequency table with absolute and percentage frequencies for one or more than one variable. By default, the function displays the absolute and percentage frequencies when specifying one variable, while the function displays only the absolute frequencies when specifying more than one variable.

Usage

```
freq(data, ..., print = c("no", "all", "perc", "v.perc"), freq = TRUE,
     split = FALSE, labels = TRUE, val.col = FALSE, round = 3, exclude = 15,
     digits = 2, as.na = NULL, write = NULL, append = TRUE, check = TRUE,
     output = TRUE)
```

Arguments

`data`	a vector, factor, or data frame.
`...`	an expression indicating the variable names in `data`, e.g., `freq(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`print`	a character string indicating which percentage(s) to be printed on the console, i.e., no percentages (`"no"`), all percentages (`"all"`), percentage frequencies (`"perc"`), and valid percentage frequencies (`"v.perc"`). Default setting when specifying one variable is `print = "all"`, while default setting when specifying more than one variable is `print = "no"` unless `split = TRUE`.
`freq`	logical: if `TRUE` (default), absolute frequencies will be shown on the console.
`split`	logical: if `TRUE`, output table is split by variables when specifying more than one variable in `...`.
`labels`	logical: if `TRUE` (default), labels for the factor levels will be used.
`val.col`	logical: if `TRUE`, values are shown in the columns, variables in the rows.
`round`	an integer value indicating the number of decimal places to be used for rounding numeric variables.
`exclude`	an integer value indicating the maximum number of unique values for variables to be included in the analysis when specifying more than one variable i.e., variables with the number of unique values exceeding `exclude` will be excluded from the analysis. It is also possible to specify `exclude = FALSE` to include all variables in the analysis.
`digits`	an integer value indicating the number of decimal places to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

The function displays valid percentage frequencies only in the presence of missing values and excludes variables with all values missing from the analysis. Note that it is possible to mix numeric variables, factors, and character variables in the data frame specified in the argument `data`. By default, numeric variables are rounded to three digits before computing the frequency table.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`result`	data frame with absolute frequencies and percentages or list with result tables, i.e., `freq` for absolute frequencies, `perc` for percentages, and `v.perc` for valid percentages

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`write.result`, `crosstab`, `descript`, `multilevel.descript`, `na.descript`.

Examples

```
#-----------------------------------------------------------------------
# Frequency Table for One Variable

# Example 1a: Frequency table for 'cyl'
freq(mtcars, cyl)

# Alternative specification without using the '...' argument
freq(mtcars$cyl)

# Example 1b: Frequency table, use 3 digit for displaying percentages
freq(mtcars, cyl, digits = 3)

#-----------------------------------------------------------------------
# Frequency Table for More Than One Variable

# Example 2a: Frequency table for 'cyl', 'gear', and 'carb'
freq(mtcars, cyl, gear, carb)
```

```
# Alternative specification without using the '...' argument
freq(mtcars[, c("cyl", "gear", "carb")])

# Example 2b: Frequency table, with percentage frequencies
freq(mtcars, cyl, gear, carb, print = "all")

#———————————————————————————————————————————————————————————————————————
# Grouping and Split Variable

# Example 3a: Frequency table, split output table
freq(mtcars, cyl, gear, carb, split = TRUE)

# Example 3b: Frequency table, exclude variables with more than 5 unique values
freq(mtcars, exclude = 5)

## Not run:
#———————————————————————————————————————————————————————————————————————
# Write Results

# Example 4a: Write Results into a text file
freq(mtcars, cyl, gear, carb, split = TRUE, write = "Frequencies.txt")

# Example 4b: Write Results into an Excel file
freq(mtcars, cyl, gear, carb, split = TRUE, write = "Frequencies.xlsx")

## End(Not run)
```

`indirect` *Confidence Intervals for the Indirect Effect*

Description

This function computes confidence intervals for the indirect effect based on the asymptotic normal method, distribution of the product method and the Monte Carlo method. By default, the function uses the Monte Carlo method for computing the two-sided 95% asymmetric confidence intervals for the indirect effect product of coefficient estimator $\hat{a}\hat{b}$.

Usage

```
indirect(a, b, se.a, se.b, print = c("all", "asymp", "dop", "mc"),
         se = c("sobel", "aroian", "goodman"), nrep = 100000,
         alternative = c("two.sided", "less", "greater"), seed = NULL,
         conf.level = 0.95, digits = 3, write = NULL, append = TRUE,
         check = TRUE, output = TRUE)
```

Arguments

a	a numeric value indicating the coefficient a, i.e., effect of X on M.

b	a numeric value indicating the coefficient b, i.e., effect of M on Y adjusted for X.
se.a	a positive numeric value indicating the standard error of a.
se.b	a positive numeric value indicating the standard error of b.
print	a character vector indicating which confidence intervals (CI) to show on the console, i.e. "all" for all CIs, "asymp" for the CI based on the asymptotic normal method, "dop" for the CI based on the distribution of the product method, and "mc" (default) for the CI based on the Monte Carlo method.
se	a character string indicating which standard error (SE) to compute for the asymptotic normal method, i.e., "sobel" for the approximate standard error by Sobel (1982) using the multivariate delta method based on a first order Taylor series approximation, "aroian" (default) for the exact standard error by Aroian (1947) based on a first and second order Taylor series approximation, and "goodman" for the unbiased standard error by Goodman (1960).
nrep	an integer value indicating the number of Monte Carlo repetitions.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
seed	a numeric value specifying the seed of the random number generator when using the Monte Carlo method.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
digits	an integer value indicating the number of decimal places to be used for displaying
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

In statistical mediation analysis (MacKinnon & Tofighi, 2013), the indirect effect refers to the effect of the independent variable X on the outcome variable Y transmitted by the mediator variable M. The magnitude of the indirect effect ab is quantified by the product of the the coefficient a (i.e., effect of X on M) and the coefficient b (i.e., effect of M on Y adjusted for X). In practice, researchers are often interested in confidence limit estimation for the indirect effect. This function offers three different methods for computing the confidence interval for the product of coefficient estimator $\hat{a}\hat{b}$:

Asymptotic Normal Method In the asymptotic normal method, the standard error for the product of the coefficient estimator $\hat{a}\hat{b}$ is computed which is used to create a symmetrical confidence interval based on the z-value of the standard normal (z) distribution assuming that the indirect effect is normally distributed. Note that the function provides three formulas for computing the standard error by specifying the argument se:

`"sobel"`: Approximate standard error by Sobel (1982) using the multivariate delta method based on a first order Taylor series approximation:

$$\sqrt{(}a^2\sigma_a^2 + b^2\sigma_b^2)$$

`"aroian"`: Exact standard error by Aroian (1947) based on a first and second order Taylor series approximation:

$$\sqrt{(}a^2\sigma_a^2 + b^2\sigma_b^2 + \sigma_a^2\sigma_b^2)$$

`"goodman"`: Unbiased standard error by Goodman (1960):

$$\sqrt{(}a^2\sigma_a^2 + b^2\sigma_b^2 - \sigma_a^2\sigma_b^2)$$

Note that the unbiased standard error is often negative and is hence undefined for zero or small effects or small sample sizes.

The asymptotic normal method is known to have low statistical power because the distribution of the product $\hat{a}\hat{b}$ is not normally distributed. (Kisbu-Sakarya, MacKinnon, & Miocevic, 2014). In the null case, where both random variables have mean equal to zero, the distribution is symmetric with kurtosis of six. When the product of the means of the two random variables is nonzero, the distribution is skewed (up to a maximum value of $\pm$ 1.5) and has a excess kurtosis (up to a maximum value of 6). However, the product approaches a normal distribution as one or both of the ratios of the means to standard errors of each random variable get large in absolute value (MacKinnon, Lockwood & Williams, 2004).

Distribution of the product method The distribution of the product method (MacKinnon et al., 2002) relies on an analytical approximation of the distribution of the product of two normally distributed variables. The method uses the standardized a and b coefficients to compute ab and then uses the critical values for the distribution of the product (Meeker, Cornwell, & Aroian, 1981) to create asymmetric confidence intervals. The distribution of the product approaches the gamma distribution (Aroian, 1947). The analytical solution for the distribution of the product is provided by the Bessel function used to the solution of differential equations and is approximately proportional to the Bessel function of the second kind with a purely imaginary argument (Craig, 1936).

Monte Carlo Method The Monte Carlo (MC) method (MacKinnon et al., 2004) relies on the assumption that the parameters a and b have a joint normal sampling distribution. Based on the parametric assumption, a sampling distribution of the product ab using random samples with population values equal to the sample estimates $\hat{a}$, $\hat{b}$, $\hat{\sigma}_a$, and $\hat{\sigma}_b$ is generated. Percentiles of the sampling distribution are identified to serve as limits for a $100(1-\alpha)\%$ asymmetric confidence interval about the sample $\hat{a}\hat{b}$ (Preacher & Selig, 2012). Note that parametric assumptions are invoked for $\hat{a}$ and $\hat{b}$, but no parametric assumptions are made about the distribution of $\hat{a}\hat{b}$.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the input specified in `a` `b`, `se.a`, and `se.b`
`args`	specification of function arguments
`result`	list with result tables, i.e., `asymp` with CI based on the asymptotic normal method, `dop` with CI based on the distribution of the product method, and `mc` for CI based on the Monte Carlo method

Note

The function was adapted from the `medci()` function in the **RMediation** package by Davood Tofighi and David P. MacKinnon (2016).

Author(s)

Takuya Yanagida

References

Aroian, L. A. (1947). The probability function of the product of two normally distributed variables. *Annals of Mathematical Statistics, 18*, 265-271. https://doi.org/10.1214/aoms/1177730442

Craig,C.C. (1936). On the frequency function of xy. *Annals of Mathematical Statistics, 7*, 1–15. https://doi.org/10.1214/aoms/1177732541

Goodman, L. A. (1960). On the exact variance of products. *Journal of the American Statistical Association, 55*, 708-713. https://doi.org/10.1080/01621459.1960.10483369

Kisbu-Sakarya, Y., MacKinnon, D. P., & Miocevic M. (2014). The distribution of the product explains normal theory mediation confidence interval estimation. *Multivariate Behavioral Research, 49*, 261–268. https://doi.org/10.1080/00273171.2014.903162

MacKinnon, D. P., Lockwood, C. M., Hoffman, J. M., West, S. G., & Sheets, V. (2002). Comparison of methods to test mediation and other intervening variable effects. *Psychological Methods, 7*, 83–104. https://doi.org/10.1037/1082-989x.7.1.83

MacKinnon, D. P., Lockwood, C. M., & Williams, J. (2004). Confidence limits for the indirect effect: Distribution of the product and resampling methods. *Multivariate Behavioral Research, 39*, 99-128. https://doi.org/10.1207/s15327906mbr3901_4

MacKinnon, D. P., & Tofighi, D. (2013). Statistical mediation analysis. In J. A. Schinka, W. F. Velicer, & I. B. Weiner (Eds.), *Handbook of psychology: Research methods in psychology* (pp. 717-735). John Wiley & Sons, Inc..

Meeker, W. Q., Jr., Cornwell, L. W., & Aroian, L. A. (1981). The product of two normally distributed random variables. In W. J. Kennedy & R. E. Odeh (Eds.), *Selected tables in mathematical statistics* (Vol. 7, pp. 1–256). Providence, RI: American Mathematical Society.

Preacher, K. J., & Selig, J. P. (2012). Advantages of Monte Carlo confidence intervals for indirect effects. *Communication Methods and Measures, 6*, 77–98. http://dx.doi.org/10.1080/19312458.2012.679848

Sobel, M. E. (1982). Asymptotic confidence intervals for indirect effects in structural equation models. In S. Leinhardt (Ed.), *Sociological methodology 1982* (pp. 290-312). Washington, DC: American Sociological Association.

Tofighi, D. & MacKinnon, D. P. (2011). RMediation: An R package for mediation analysis confidence intervals. *Behavior Research Methods, 43*, 692-700. https://doi.org/10.3758/s13428-011-0076-x

See Also

`multilevel.indirect`

Examples

```
# Example 1: Monte Carlo Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18)

# Example 2: Distribution of the Product Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18, print = "dop")

# Example 3: Asymptotic Normal Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18, print = "asymp")

## Not run:
# Example 4: Write results into a text file
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18, write = "Indirect.txt")
## End(Not run)
```

`item.alpha` *Coefficient Alpha, Hierarchical Alpha, and Ordinal Alpha*

Description

This function computes point estimate and confidence interval for the coefficient alpha (aka Cronbach's alpha) and ordinal coefficient alpha (aka categorical alpha) along with corrected item-total correlations or standardized factor loadings and coefficient alphas if the item is deleted. By default, the function computes the formula-based coefficient alpha using pairwise deletion in the presence of missing data.

Usage

```
item.alpha(data, ..., rescov = NULL, ordered = FALSE, exclude = NULL,
           correct = TRUE, std = FALSE,
           estimator = c("ML", "GLS", "WLS", "DWLS", "ULS", "PML"),
           missing = c("listwise", "pairwise", "fiml"),
           print = c("all", "alpha", "item"), digits = 2, r.digits = 3,
           conf.level = 0.95, as.na = NULL, write = NULL, append = TRUE,
           check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame. Note that at least two items are needed for computing coefficient alpha
`...`	an expression indicating the variable names in `data` e.g., `item.alpha(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`rescov`	a character vector or a list of character vectors for specifying residual covariances when computing coefficient alpha, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2` or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.

ordered	logical: if TRUE, variables are treated as ordered (ordinal) variables to compute ordinal coefficient alpha.
exclude	a character vector indicating items to be excluded from the analysis.
correct	logical: if TRUE (default), the corrected item-total correlation is computed.
std	logical: if TRUE, the standardized coefficient omega is computed.
estimator	a character string indicating the estimator to be used (see 'Details' in the item.cfa function) when specifying residual covariances using the rescov argument or when using full information maximum likelihood method for missing data handling, i.e., missing = "fiml". By default, "ULS" is used for computing coefficient alpha in the presence of residual covariances. Note that the argument estimator switches to "ML" when specifying missing = "fiml".
missing	a character string indicating how to deal with missing data. (see 'Details' in the item.cfa function). By default, pairwise deletion (missing = "pairwise") is used for computing coefficient alpha and ordinal coefficient alpha. Full information maximum likelihood (FIML) method is available for estimating coefficient alpha and is requested by specifying missing = "fiml" along with estimator = "ML". Note that FIML method is not available for computing ordinal coefficient alpha.
print	a character vector indicating which results to show, i.e. "all" for all results "alpha" (default) for the coefficient alpha, and "item" for item statistics.
digits	an integer value indicating the number of decimal places to be used for displaying mean, standard deviation, minimum, and maximum.
r.digits	an integer value indicating the number of decimal places to be used for displaying alpha and standardized factor loadings.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Details

Coefficient Alpha This function computes the coefficient alpha using either a formula-based method or a confirmatory factor analysis (CFA). The latter conducts a CFA based on the essentially tau-equivalent measurement model (Graham, 2006) using the cfa() function in the **lavaan** package by Yves Rosseel (2019). By default, the function employs the formula-based method and uses listwise deletion to handle missing values. The function switches to the CFA-based method when residual covariances are specified using the rescov argument, when full information maximum likelihood (FIML) method is requested for missing data handling by

specifying `missing = "fiml"`, or when the `estimator` argument is set to any other estimation method other than the default estimator `ULS`.

Ordinal Coefficient Alpha The ordinal coefficient alpha (Zumbo et al., 2007) is calculated by applying the formula for coefficient alpha to the polychoric correlation matrix, rather than to the the variance-covariance or product-moment correlation matrix. The ordinal coefficient alpha should be interpreted only as a hypothetical estimate of an alternative reliability, where a test's ordinal categorical response options have been modified to include an infinite number of response options and concludes that coefficient alpha should not be reported as a measure of a test's reliability. However, Zumbo and Kroc (2019) argued that Chalmers' critique of ordinal coefficient alpha is unfounded, and that ordinal coefficient alpha may be the most appropriate quantifier of reliability when using Likert-type measurement to study a latent continuous random variable.

Confidence Interval The confidence interval for the (ordinal) coefficient alpha is computed using the procedure by Feldt et al. (1987). Note that there are at least 10 other procedures for computing the confidence interval (see Kelley and Pornprasertmanit, 2016), which are implemented in the `ci.reliability()` function in the **MBESSS** package by Ken Kelley (2019).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`model.fit`	fitted lavaan object (`mod.fit`)
`result`	list with result tables, i.e., `alpha` for a table with coefficient alpha and `itemstat` for a table with item statistics

Note

Computation of the polyserial correlation coefficient is based on the `polyserial()` function in the **polycor** package by John Fox (2025)

Author(s)

Takuya Yanagida

References

Chalmers, R. P. (2018). On misconceptions and the limited usefulness of ordinal alpha. *Educational and Psychological Measurement, 78*, 1056-1071. https://doi.org/10.1177/0013164417727036

Cronbach, L.J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297-334. https://doi.org/10.1007/BF02310555

Cronbach, L.J. (2004). My current thoughts on coefficient alpha and successor procedures. *Educational and Psychological Measurement, 64*, 391-418. https://doi.org/10.1177/0013164404266386

Feldt, L. S., Woodruff, D. J., & Salih, F. A. (1987). Statistical inference for coefficient alpha. *Applied Psychological Measurement*, 11 93-103. https://doi.org/10.1177/014662168701100107

Fox, J. (2025). *polycor: Polychoric and polyserial correlations*. R package version 0.8-2. https://doi.org/10.32614/CRAN.package.polycor

Graham, J. M. (2006). Congeneric and (essentially) tau-equivalent estimates of score reliability: What they are and how to use them. *Educational and Psychological Measurement, 66*(6), 930–944. https://doi.org/10.1177/0013164406288165

Kelley, K., & Pornprasertmanit, S. (2016). Confidence intervals for population reliability coefficients: Evaluation of methods, recommendations, and software for composite measures. *Psychological Methods, 21*, 69-92. https://doi.org/10.1037/a0040086.

Ken Kelley (2019). *MBESS: The MBESS R Package*. R package version 4.6.0. https://CRAN.R-project.org/package=MBESS

Zumbo, B. D., & Kroc, E. (2019). A measurement is a choice and Stevens' scales of measurement do not help make it: A response to Chalmers. *Educational and Psychological Measurement, 79*, 1184-1197. https://doi.org/10.1177/0013164419844305

Zumbo, B. D., Gadermann, A. M., & Zeisser, C. (2007). Ordinal versions of coefficients alpha and theta for Likert rating scales. *Journal of Modern Applied Statistical Methods, 6*, 21-29. https://doi.org/10.22237/jmasm/1177992180

See Also

item.omega, item.cfa, item.invar, item.reverse, item.scores, write.result

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#-----------------------------------------------------------------------
# Continuous Data

# Example 1a: Coefficient alpha, listwise deletion
item.alpha(HolzingerSwineford1939, x1::x9)

# Example 1b: Full information maximum likelihood method
item.alpha(HolzingerSwineford1939, x1::x9, estimator = "ML", missing = "fiml")

# Example 2: Coefficient alpha and item statistics after excluding 'x3'
item.alpha(HolzingerSwineford1939, x1::x9, exclude = "x3", print = "all")

# Example 3a: Residual covariance between 'x1' and 'x2'
item.alpha(HolzingerSwineford1939, x1::x9, rescov = c("x1", "x2"))

# Example 3b: Residual covariances between 'x1' and 'x2', and 'x2' and 'x3'
item.alpha(HolzingerSwineford1939, x1::x9, rescov = list(c("x1", "x2"), c("x2", "x3")))

# Example 4: Summary of the CFA model used to compute coefficient alpha
```

```
lavaan::summary(item.alpha(HolzingerSwineford1939, x1::x9, output = FALSE)$model.fit,
                standardized = TRUE)

#————————————————————————————————————————————————————————————————————————
# Polytomous Data

# Example 5: Ordinal coefficient alpha and item statistics
item.alpha(data.items, pitem1, pitem2r, pitem3r, pitem4::pitem6, type = "categ",
           print = "all")

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 6a: Write Results into a text file
item.alpha(HolzingerSwineford1939, x1::x9, print = "all", write = "Alpha.txt")

# Example 6b: Write Results into an Excel file
item.alpha(HolzingerSwineford1939, x1::x9, print = "all", write = "Alpha.xlsx")

## End(Not run)
```

---

item.cfa *Confirmatory Factor Analysis*

Description

This function is a wrapper function for conducting confirmatory factor analysis with continuous and/or ordered-categorical indicators by calling the `cfa` function in the R package **lavaan**. By default, the function provides a table with univariate sample statistics, model fit information, and parameter estimates. Additionally, variance-covariance coverage of the data, modification indices, residual correlation matrix, and relative Opdyke distribution percentile matrix can be requested by specifying the argument `print`.

Usage

```
item.cfa(data, ..., model = NULL, rescov = NULL, hierarch = FALSE,
         meanstructure = TRUE, ident = c("marker", "var", "effect"),
        parameterization = c("delta", "theta"), ordered = FALSE, cluster = NULL,
         estimator = c("ML", "MLM", "MLMV", "MLMVS", "MLF", "MLR",
                       "GLS", "WLS", "DWLS", "WLSM", "WLSMV", "WLSMVS",
                       "ULS", "ULSM", "ULSMV", "ULSMVS", "DLS", "PML"),
         test = c("none", "standard", "satorra.bentler", "scaled.shifted",
                  "mean.var.adjusted", "yuan.bentler", "yuan.bentler.mplus",
                  "browne.residual.adf", "browne.residual.nt",
                  "mean.var.adjusted.corrected", "scaled.shifted.corrected"),
         se = c("none", "standard", "robust.sem", "robust.huber.white"),
         missing = c("listwise", "pairwise", "fiml", "two.stage",
                     "robust.two.stage", "doubly.robust"),
         print = c("all", "summary", "coverage", "descript", "fit", "est",
```

```
            "modind", "resid", "opdyke"),
mod.minval = 6.63, resid.minval = 0.1, opdyke.prec = 1,
opdyke.minmax = c(0.40, 0.60), color = "b.red",
style = c("regular", "bold", "italic"), digits = 3, p.digits = 3,
as.na = NULL, write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. If `model = NULL`, confirmatory factor analysis based on a measurement model with one factor labeled `f` comprising all variables in the data frame is conducted. Note that the cluster variable is excluded from `data` when specifying `cluster`. If `model` is specified, the data frame needs to contain all variables used in the argument `model` and the cluster variable when specifying `cluster`.

`...` an expression indicating the variable names in `data`, e.g., `item.cfa(x1, x2, x3, data = dat)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`model` a character vector specifying a measurement model with one factor, or a list of character vectors for specifying a measurement model with more than one factor, e.g., `model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `f` comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"),factor2 = c("x5", "x6", "x7", "x8"))` for specifying a measurement model with two latent factors labeled `factor1` and `factor2` each comprising four indicators. Note that the name of each list element is used to label factors, i.e., all list elements need to be named, otherwise factors are labeled with `"f1"`, `"f2"`, `"f3"` and so on.

`rescov` a character vector or a list of character vectors for specifying residual covariances, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2`, or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.

`hierarch` logical: if `TRUE`, a second-order factor model is specified given at least three first-order factors were specified in `model`. Note that it is not possible to specify more than one second-order factor.

`meanstructure` logical: if `TRUE` (default), intercept/means of observed variables means of latent variables will be added to the model. Note that `meanstructure = FALSE` is only applicable when the `missing` is `listwise`, `pairwise`, or `doubly-robust`.

`ident` a character string indicating the method used for identifying and scaling latent variables, i.e., `"marker"` for the marker variable method fixing the first factor loading of each latent variable to 1, `"var"` for the fixed variance method fixing the variance of each latent variable to 1, or `"effect"` for the effects-coding method using equality constraints so that the average of the factor loading for each latent variable equals 1. By default, fixed variance method is used when `hierarch = FALSE`, whereas marker variable method is used when `hierarch = TRUE`.

`parameterization` a character string indicating the method used for identifying and scaling latent

variables when indicators are ordered, i.e., "delta" (default) for delta parameterization and "theta" for theta parameterization.

ordered — if FALSE (default), all indicators of the measurement model are treated as continuous. If TRUE, all indicators of the measurement model are treated as ordered (ordinal). Alternatively, a character vector indicating which variables to treat as ordered (ordinal) variables can be specified.

cluster — either a character string indicating the variable name of the cluster variable in data, or a vector representing the nested grouping structure (i.e., group or cluster variable) for computing cluster-robust standard errors. Note that cluster-robust standard errors are not available when treating indicators of the measurement model as ordered (ordinal).

estimator — a character string indicating the estimator to be used (see 'Details'). By default, "MLR" is used for CFA models with continuous indicators (i.e., ordered = FALSE) and "WLSMV" is used for CFA model with ordered-categorical indicators (i.e., ordered = TRUE).

test — a character string indicating the test statistics, i.e., "none" for no test statistic, "standard" for a conventional chi-square test, "satorra.bentler" for a Satorra-Bentler scaled test statistic, "scaled.shifted" for a mean and variance adjusted test statistic (scaled and shifted), "mean.var.adjusted" for a mean and variance adjusted test statistic (Satterthwaite style), "yuan.bentler" for a Yuan-Bentler scaled test statistic, "yuan.bentler.mplus" for a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic, "browne.residual.adf" for a Browne's residual-based test statistic using ADF theory, "browne.residual.nt" for Browne's residual-based test statistic using normal theory, "mean.var.adjusted.corrected" and "scaled.shifted.corrected" for a test statistic with the corrected (unbiased) estimator of the trace of the squared UGamma matrix (Hayakawa, 2019), which remains accurate when the number of observed variables is large relative to the sample size. Note that the default setting is depending on the argument estimator and the argument cluster (see 'Details').

se — a character string indicating the standard errors, i.e., "none" for no standard errors, "standard" for conventional standard error based on inverting the expected observed or first.order information matrix, "robust.sem" for conventional robust standard errors, and "robust.huber.white" for the 'MLR' (aka pseudo ML, Huber-White) approach. The argument "se" is set to "two.stage" or "robust.two.stage" when specifying missing = "two.stage" or missing = "robust.two.stage". Note that the default setting is depending on the argument estimator and the argument cluster (see 'Details').

missing — a character string indicating how to deal with missing data, i.e., "listwise" for listwise deletion, "pairwise" for pairwise deletion, "fiml" for full information maximum likelihood method, "two.stage" for two-stage maximum likelihood method, "robust.two.stage" for robust two-stage maximum likelihood method, and "doubly-robust" for doubly-robust method (see 'Details'). By default, "fiml" is used for CFA models with continuous indicators when using estimator = "MLR", and "pairwise" for CFA models with ordered-categorical indicators when using estimator = "pairwise".

`print` a character vector indicating which results to show on the console, i.e. `"all"` for all results, `"summary"` for a summary of the specification of the estimation method and missing data handling in lavaan, `"coverage"` for the variance-covariance coverage of the data, `"descript"` for descriptive statistics, `"fit"` for model fit, `"est"` for parameter estimates, `"modind"` for modification indices, `"resid"` for the residual correlation, and `"opdyke"` for the discrepancy between model-implied and observed correlation expressed in terms of the relative percentile of an Opdyke distribution (McNeish, 2025). By default, a summary of the specification, model fit, and parameter estimates are printed. Note that `"fit"` will be excluded when specifying the argument `test = "none"`.

`mod.minval` numeric value to filter modification indices and only show modifications with a modification index value equal or higher than this minimum value. By default, modification indices equal or higher 6.63 are printed. Note that a modification index value of 6.63 is equivalent to a significance level of $\alpha = .01$.

`resid.minval` numeric value indicating the minimum absolute residual correlation coefficients and standardized means to highlight. By default, absolute residual correlation coefficients and standardized means equal or higher 0.1 are highlighted. Note that highlighting can be disabled by setting the minimum value to 1 or by setting the argument `color` to `"default"`.

`opdyke.prec` a numeric value indicating the precision of the probability density function calculations of the Opdyke distribution. The default is `1` which calculates the PDF for polar angles between $(0, pi)$ in 0.01 increments. Specifying `10` calculates the PDF polar angles between $(0, pi)$ in 0.001 increments, which takes considerably longer, especially if there are many correlation elements.

`opdyke.minmax` a numeric vector with two elements indicating the minimum and maximum percentile of the Opdyke distribution that is considered to be acceptably close to the observed correlation represented by the Opdyke distribution median. Predicted correlation outside the range will be color highlighted in line with to the argument `color`.

`color` a character string indicating the text color for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval` and predicted correlations outside the minimum and maximum percentile of the Opdyke distribution, i.e., i.e., `"default"` for the default text color without color coding and various text colors for highlighting `"red"`, `"b.red"` (default), `"green"`, `"b.green"`, `"blue"`, or `"b.blue"`, see the help page of the `chr.color` function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to `"default"` when using R Markdown and when writing the output into a text file (`.txt`).

`style` a character vector indicating the font style for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval`, i.e., `"regular"` (default) for regular text, `"bold"` for bold text, and `"italic"` for italic text. Note that the font style `"bold"` and `"italic"` can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument `color` needs to be specified to change the style of the text, e.g. `color = "black"` and `style = "bold"` to for bold text.

`digits` an integer value indicating the number of decimal places to be used for displaying results.

p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data but not to cluster.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Details

Estimator The R package **lavaan** provides seven estimators that affect the estimation, namely "ML", "GLS", "WLS", "DWLS", "ULS", "DLS", and "PML". All other options for the argument estimator combine these estimators with various standard error and chi-square test statistic computation. Note that the estimators also differ in how missing values can be dealt with (e.g., listwise deletion, pairwise deletion, or full information maximum likelihood, FIML).

- "ML": Maximum likelihood with conventional standard errors (se = "standard") and conventional test statistic (test = "standard"). For both complete and incomplete data using pairwise deletion or FIML.
- "MLM": Maximum likelihood parameter estimates with conventional robust standard errors (se = "robust.sem") and a Satorra-Bentler scaled test statistic (test = "satorra.bentler") that are robust to non-normality. For complete data only.
- "MLMV": Maximum likelihood parameter estimates with conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "scaled.shifted") using a scale-shifted approach that are robust to non-normality. For complete data only.
- "MLMVS": Maximum likelihood parameter estimates with conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "mean.var.adjusted") using the Satterthwaite approach that are robust to non-normality. For complete data only.
- "MLF": Maximum likelihood parameter estimates with standard errors approximated by first-order derivatives (se = "standard") and conventional test statistic (test = "standard"). For both complete and incomplete data using pairwise deletion or FIML.
- "MLR": Maximum likelihood parameter estimates with Huber-White robust standard errors (se = "robust.huber.white") a test statistic which is asymptotically equivalent to the Yuan-Bentler T2* test statistic (test = "yuan.bentler.mplus") that are robust to non-normality and non-independence of observed when specifying a cluster variable using the argument cluster. For both complete and incomplete data using pairwise deletion or FIML.
- "GLS": Generalized least squares parameter estimates with conventional standard errors (se = "standard") and conventional test statistic (test = "standard") that uses a normal-theory based weight matrix. For complete data only.

- "WLS": Weighted least squares parameter estimates (sometimes called ADF estimation) with conventional standard errors (se = "standard") and conventional test statistic that uses a full weight matrix (test = "standard"). For both complete and incomplete data using pairwise deletion.
- "DWLS": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation with conventional standard errors (se = "standard") and conventional test statistic (test = "standard"). For both complete and incomplete data using pairwise deletion.
- "WLSM": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation, but uses the full weight matrix for computing the conventional robust standard errors (se = "robust.sem") and a Satorra-Bentler scaled test statistic (test = "satorra.bentler"). For both complete and incomplete data using pairwise deletion.
- "WLSMV": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation, but uses the full weight matrix for computing the conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "scaled.shifted") using a scale-shifted approach. For both complete and incomplete data using pairwise deletion.
- "WLSMVS": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation, but uses the full weight matrix for computing the conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "mean.var.adjusted") using the Satterthwaite approach. For both complete and incomplete data using pairwise deletion.
- "ULS": Unweighted least squares parameter estimates with conventional standard errors (se = "standard") and conventional test statistic (test = "standard"). For both complete and incomplete data using pairwise deletion.
- "ULSM": Unweighted least squares parameter estimates with conventional robust standard errors (se = "robust.sem") and a Satorra-Bentler scaled test statistic (test = "satorra.bentler"). For both complete and incomplete data using pairwise deletion.
- "ULSMV": Unweighted least squares parameter estimates with conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "scaled.shifted") using a scale-shifted approach. For both complete and incomplete data using pairwise deletion.
- "ULSMVS": Unweighted least squares parameter estimates with conventional robust standard errors (se = "robust.sem") and a mean and a variance adjusted test statistic (test = "mean.var.adjusted") using the Satterthwaite approach. For both complete and incomplete data using pairwise deletion.
- "DLS": Distributionally-weighted least squares parameter estimates with conventional robust standard errors (se = "robust.sem") and a Satorra-Bentler scaled test statistic (test = "satorra.bentler"). For complete data only.
- "PML": Pairwise maximum likelihood parameter estimates with Huber-White robust standard errors (se = "robust.huber.white") and a mean and a variance adjusted test statistic (test = "mean.var.adjusted") using the Satterthwaite approach. For both complete and incomplete data using pairwise deletion.

Missing Data The R package **lavaan** provides six methods for dealing with missing data:

- "listwise": Listwise deletion, i.e., all cases with missing values are removed from the data before conducting the analysis. This is only valid if the data are missing completely

at random (MCAR).

- "pairwise": Pairwise deletion, i.e., each element of a variance-covariance matrix is computed using cases that have data needed for estimating that element. This is only valid if the data are missing completely at random (MCAR).
- "fiml": Full information maximum likelihood (FIML) method, i.e., likelihood is computed case by case using all available data from that case. FIML method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "two.stage": Two-stage maximum likelihood estimation, i.e., sample statistics is estimated using EM algorithm in the first step. Then, these estimated sample statistics are used as input for a regular analysis. Standard errors and test statistics are adjusted correctly to reflect the two-step procedure. Two-stage method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "robust.two.stage": Robust two-stage maximum likelihood estimation, i.e., two-stage maximum likelihood estimation with standard errors and a test statistic that are robust against non-normality. Robust two-stage method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "doubly.robust": Doubly-robust method only applicable for pairwise maximum likelihood estimation (i.e., estimator = "PML").

Convergence and Model Idenfitification Checks In line with the R package **lavaan**, this functions provides several checks for model convergence and model identification:

- Degrees of freedom: An error message is printed if the number of degrees of freedom is negative, i.e., the model is not identified.
- Model convergence: An error message is printed if the optimizer has not converged, i.e., results are most likely unreliable.
- Standard errors: An error message is printed if the standard errors could not be computed, i.e., the model might not be identified.
- Variance-covariance matrix of the estimated parameters: A warning message is printed if the variance-covariance matrix of the estimated parameters is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.
- Negative variances of observed variables: A warning message is printed if the estimated variances of the observed variables are negative.
- Variance-covariance matrix of observed variables: A warning message is printed if the estimated variance-covariance matrix of the observed variables is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.
- Negative variances of latent variables: A warning message is printed if the estimated variances of the latent variables are negative.
- Variance-covariance matrix of latent variables: A warning message is printed if the estimated variance-covariance matrix of the latent variables is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.

Note that unlike the R package **lavaan**, the item.cfa function does not provide any results when the degrees of freedom is negative, the model has not converged, or standard errors could not be computed.

Model Fit The item.cfa function provides the chi-square test, incremental fit indices (i.e., CFI and TLI), and absolute fit indices (i.e., RMSEA, and SRMR) to evaluate overall model fit. However, different versions of the CFI, TLI, and RMSEA are provided depending on the

estimator. Unlike the R package **lavaan**, the different versions are labeled with `Standard`, `Scaled`, and `Robust` in the output:

- `"Standard"`: CFI, TLI, and RMSEA without any non-normality corrections. These fit measures based on the normal theory maximum likelihood test statistic are sensitive to deviations from multivariate normality of endogenous variables. Simulation studies by Brosseau-Liard et al. (2012), and Brosseau-Liard and Savalei (2014) showed that the uncorrected fit indices are affected by non-normality, especially at small and medium sample sizes (e.g., n < 500).
- `"Scaled"`: Population-corrected robust CFI, TLI, and RMSEA with ad hoc non-normality corrections that simply replace the maximum likelihood test statistic with a robust test statistic (e.g., mean-adjusted chi-square). These fit indices change the population value being estimated depending on the degree of non-normality present in the data. Brosseau-Liard et al. (2012) demonstrated that the ad hoc corrected RMSEA increasingly accepts poorly fitting models as non-normality in the data increases, while the effect of the ad hoc correction on the CFI and TLI is less predictable with non-normality making fit appear worse, better, or nearly unchanged (Brosseau-Liard & Savalei, 2014).
- `"Robust"`: Sample-corrected robust CFI, TLI, and RMSEA with non-normality corrections based on formula provided by Li and Bentler (2006) and Brosseau-Liard and Savalei (2014). These fit indices do not change the population value being estimated and can be interpreted the same way as the uncorrected fit indices when the data would have been normal.

In conclusion, the use of sample-corrected fit indices (`Robust`) instead of population-corrected fit indices (`Scaled`) is recommended. Note that when sample size is very small (e.g., n < 200), non-normality correction does not appear to adjust fit indices sufficiently to counteract the effect of non-normality (Brosseau-Liard & Savalei, 2014).

Modification Indices The `item.cfa` function provides modification indices when requested by using the `print` argument. Modification indices (aka score tests) are univariate Lagrange Multipliers (LM) representing a chi-square statistic with a single degree of freedom. LM approximates the amount by which the chi-square test statistic would decrease if a fixed or constrained parameter is freely estimated (Kline, 2023). However, (standardized) expected parameter change (EPC) values should also be inspected since modification indices are sensitive to sample size. EPC values are an estimate of how much the parameter would be expected to change if it were freely estimated (Brown, 2023).

Residual Correlation Matrix The `item.cfa` function provides the residual correlation matrix when requested by using the `print` argument. The residual correlation matrix is computed by separately converting the sample covariance and model-implied covariance matrices to correlation matrices before calculation differences between observed and predicted covariances (i.e., `type = "cor.bollen"`). As a rule of thumb, absolute correlation residuals greater than .10 indicate possible evidence for poor local fit, whereas smaller correlation residuals than 0.05 indicate negligible degree of model misfit (Maydeu-Olivares, 2017). There is no reliable connection between the size of diagnostic statistics (i.e., modification indices and residuals) and the type or amount of model misspecification since (1) diagnostic statistics are themselves affected by misspecification, (2) misspecification in one part of the model distorts estimates in other parts of the model (i.e., error propagation), and (3) equivalent models have identical residuals but contradict the pattern of causal effects (Kline, 2023). Note that according to Kline (2023) "any report of the results without information about the residuals is deficient" (p. 172).

Relative Opdyke Distribution Percentile Matrix The `item.cfa` function provides the relative Opdyke

distribution percentile matrix (McNeish, 2025) when requested by using the `print` argument. Approximate local fit evaluation based on the relative Opdyke distribution percentile does not rely on a fixed interval, such as the common $r \pm 0.10$ rule of thumb for interpreting correlation residuals. Instead, it is based on the percentiles of the distribution of values that an observed correlation could take while still yielding a valid correlation matrix. The Opdyke distribution for the observed correlation can determine the percentile rank associated with the model-implied correlation. Accordingly, approximate local fit can be evaluated by whether the model-implied correlation falls within a certain percentile of the observed correlation's distribution, rather than using a heuristic fixed interval. The observed correlation corresponds to the median of the Opdyke distribution, i.e., approximate fit can be gauged by proximity to the 50th percentile of the distribution. In other words, correlation discrepancies are expressed in terms of relative percentiles of the Opdyke distribution. Note that the common $r \pm 0.10$ rule of thumb for interpreting correlation residuals roughly corresponds to a model-implied correlation between the 40th and 60th percentiles of the observed correlation's Opdyke distribution. However, this rule may not necessarily generalize to all observed correlations. Note that Opdyke percentiles are only applicable to correlation structure elements, but are not applicable to mean structure residuals.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data`
`args`	specification of function arguments
`model`	specified model
`model.fit`	fitted lavaan object (`mod.fit`)
`check`	results of the convergence and model identification check
`result`	list with result tables, i.e., `summary` for the summary of the specification of the estimation method and missing data handling in lavaan, `coverage` for the variance-covariance coverage of the data, `descript` for descriptive statistics, `fit` for model fit `est` for parameter estimates, `modind` for modification indices, `resid` for the residual correlation matrices and standardized residual means, and `opdyke` for the relative Opdyke distribution percentile matrix

Note

The function uses the functions `cfa`, `lavInspect`, `lavTech`, `modindices`, `parameterEstimates`, and `standardizedsolution` provided in the R package **lavaan** by Yves Rosseel (2012). The relative Opdyke distribution percentile matrix is based on the functions `opdyke` and `opdyke.percentiles` provided in the R package **opddyke** by Dan McNeish (2026).

Author(s)

Takuya Yanagida

References

Brosseau-Liard, P. E., Savalei, V., & Li. L. (2012). An investigation of the sample performance of two nonnormality corrections for RMSEA, *Multivariate Behavioral Research, 47*, 904-930. https://doi.org/10.1080/00273171.2014.933697

Brosseau-Liard, P. E., & Savalei, V. (2014) Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research, 49*, 460-470. https://doi.org/10.1080/00273171.2014.933697

Brown, T. A. (2023). Confirmatory factor analysis. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (2nd ed.) (pp. 361–379). The Guilford Press.

Hayakawa K. (2019). Corrected goodness-of-fit test in covariance structure analysis. *Psychological Methods, 24*(3), 371-389. https://doi.org/10.1037/met0000180

Kline, R. B. (2023). *Principles and practice of structural equation modeling* (5th ed.). Guilford Press.

Li, L., & Bentler, P. M. (2006). Robust statistical tests for evaluating the hypothesis of close fit of misspecified mean and covariance structural models. *UCLA Statistics Preprint #506*. University of California.

Maydeu-Olivares, A. (2017). Assessing the size of model misfit in structural equation models. *Psychometrika, 82*(3), 533–558. https://doi.org/10.1007/s11336-016-9552-7

McNeish, D. (2025). Less Heuristic Approximate Local Fit Evaluation in Structural Equation Models. *Structural Equation Modeling: A Multidisciplinary Journal, 32*(4), 590-605. https://doi.org/10.1080/10705511.2025.2473342

McNeish D (2026). opdyke: Opdyke Percentiles for Approximate Local Fit in Structural Equation Models. R package version 1.0.0. Retrieved from https://github.com/dmcneish18/opdyke

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

See Also

item.alpha, item.omega, item.scores

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#-----------------------------------------------------------------------
# Measurement Model with One Factor

# Example 1a: Specification using the argument '...'
item.cfa(HolzingerSwineford1939, x1:x3)

# Example 1b: Alternative specification without using the '...' argument
item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")])

# Example 1c: Alternative specification using the argument 'model'
item.cfa(HolzingerSwineford1939, model = c("x1", "x2", "x3"))
```

```
# Example 1e: Alternative specification using the argument 'model'
item.cfa(HolzingerSwineford1939, model = list(visual = c("x1", "x2", "x3")))

#------------------------------------------------------------------------
# Measurement Model with Three Factors

# Example 2: Specification using the argument 'model'
item.cfa(HolzingerSwineford1939,
         model = list(visual = c("x1", "x2", "x3"),
                      textual = c("x4", "x5", "x6"),
                      speed = c("x7", "x8", "x9")))

#------------------------------------------------------------------------
# Residual Covariances

# Example 3a: One residual covariance
item.cfa(HolzingerSwineford1939,
         model = list(visual = c("x1", "x2", "x3"),
                      textual = c("x4", "x5", "x6"),
                      speed = c("x7", "x8", "x9")),
         rescov = c("x1", "x2"))

# Example 3b: Two residual covariances
item.cfa(HolzingerSwineford1939,
         model = list(visual = c("x1", "x2", "x3"),
                      textual = c("x4", "x5", "x6"),
                      speed = c("x7", "x8", "x9")),
         rescov = list(c("x1", "x2"), c("x4", "x5")))

#------------------------------------------------------------------------
# Second-Order Factor Model based on Three First-Order Factors

# Example 4
item.cfa(HolzingerSwineford1939,
         model = list(visual = c("x1", "x2", "x3"),
                      textual = c("x4", "x5", "x6"),
                      speed = c("x7", "x8", "x9")), hierarch = TRUE)

#------------------------------------------------------------------------
# Measurement Model with Ordered-Categorical Indicators

# Example 5
item.cfa(data.items, pitem1, pitem2r, pitem3r, pitem4, ordered = TRUE)

#------------------------------------------------------------------------
# Cluster-Robust Standard Errors

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Example 6a: Specification using the '...' argument
item.cfa(y4:y6, data = Demo.twolevel, cluster = "cluster")
```

```
# Example 6b: Alternative specification without using the '...' argument
item.cfa(Demo.twolevel[, c("y4", "y5", "y6")], cluster = Demo.twolevel$cluster)

# Example 6c: Alternative specification without using the '...' argument
item.cfa(Demo.twolevel[, c("y4", "y5", "y6", "cluster")], cluster = "cluster")

#----------------------------------------------------------------------------
# Argument 'print'

# Example 7a: Request all results
item.cfa(HolzingerSwineford1939, x1::x9, print = "all")

# Example 7b: Request modification indices with value equal or higher than 20
item.cfa(HolzingerSwineford1939, x1::x9, print = "modind", mod.minval = 20)

# Example 7c: Highlight absolute residual correlation equal or higher than 0.05
item.cfa(HolzingerSwineford1939, x1::x9, print = "resid", resid.minval = 0.05,
         color = "b.blue")

# Example 7d: Highlight Opdyke distribution percentiles outside 0.45 and 0.55
item.cfa(HolzingerSwineford1939, x1::x9, print = "opdyke", opdyke.minmax = c(0.45, 0.55),
         color = "black", style = "bold")

#----------------------------------------------------------------------------
# lavaan Summary of the Estimated Model

# Example 8
mod <- item.cfa(HolzingerSwineford1939, x1, x2, x3, output = FALSE)

lavaan::summary(mod$model.fit, standardized = TRUE, fit.measures = TRUE)

#----------------------------------------------------------------------------
# Write Results

# Example 9a: Write Results into a text file
item.cfa(HolzingerSwineford1939, x1, x2, x3, print = "all", write = "CFA.txt")

# Example 9b: Write Results into an Excel file
item.cfa(HolzingerSwineford1939, x1, x2, x3, print = "all", write = "CFA.xlsx")

## End(Not run)
```

`item.dfi` *Dynamic Fit Index Cutoffs*

Description

This function computes simulation-based dynamic fit index cutoffs (McNeish & Wolf, 2022, 2023) for evaluating confirmatory factor models based on multivariate normal, multivariate non-normal, likert-type, and categorical data using the the omitted paths approach.

Usage

```
item.dfi(model, data = NULL, n = NULL, type = c("norm", "nnorm", "likert", "categ"),
         level = c(0, 1, 2, 3), res.cor = 0.3, estimator = NULL,
         fit.indices = c("standard", "scaled", "robust"),
         specific = 0.95, sensitiv = 0.95, nrep = 500, seed = TRUE,
         progress = TRUE, print = c("all", "summary", "model", "cutoff"),
         digits = 3, plot = FALSE, filename = NULL, width = NA, height = NA,
         dpi = 600, write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`model` an object of class lavaan, i.e., a fitted CFA measurement model, an object of class misty of type `item.cfa`, or a character string indicating the lavaan model syntax for a CFA measurement model.

`data` a data frame. Note that this argument is needed only when specifying a character string for the argument `model` while specifying `"nnorm"` or `"likert"` for the argument `type` as the data frame is extracted from the fitted model when specifying an object of class `lavaan` or `misty` for the argument `model`.

`n` a numeric value indicating the number of observations for simulating fit index cutoffs. Note that this argument is needed only when specifying a character string for the argument `model` as the number of observations of the fitted model is extracted from the fitted model when specifying an object of class `lavaan` or `misty` for the argument `model`.

`type` a character string indicating how data are simulated, i.e., `"norm"` (default when specifying a character string for the argument `model`) for assuming multivariate normality across all items, `"nnorm"` (default when specifying an object of class `lavaan` or `misty` for the argument `model`) for assuming multivariate non-normality across all items, `"likert"` assuming discrete likert-type items treated as continuous, or `"categ"` (default when specifying a categorical CFA model for the argument `model`) assuming ordered-categorical items.

`level` a numeric vector (default: `c(0, 1, 2, 3)`) indicating the levels of misspecification for which fit index cutoffs are simulated. Note that `0` represents the true model without any misspecification and always needs to be included in the argument `level`.

`res.cor` a numeric value (default: `0.3`) indicating the magnitude of the residual correlations between items introduced for model misspecification in a one-factor CFA model.

`estimator` a character string indicating the estimator to be used for simulating fit index cutoffs (see 'Details' in the help page of the `item.cfa()` function). Note that this argument is needed only when specifying a character string for the argument `model` as the estimator of the fitted model is extracted from the fitted model when specifying an object of class `lavaan` or `misty` for the argument `model`.

`fit.indices` a character string indicating which version of the CFI, TLI, and RMSEA to compute for simulating fit index cutoffs, i.e., `"standard"` for fit indices without any non-normality correction, `"scaled"` for population-corrected robust fit indices with ad hoc non-normality correction, and `robust` for sample-corrected robust fit indices.

specific	a numeric value (default: 0.95) indicating specificity, i.e., proportions of correct models identified by the cutoffs.
sensitiv	a numeric value (default: 0.95) indicating sensitivity, i.e., proportions of incorrect models identified by the cutoffs.
nrep	an integer value (default: 500) indicating the number of replications in simulating fit index cutoffs.
seed	logical: if TRUE (default), the same seed of the pseudo-random numbers for simulating fit index cutoffs are used as in the R package **dynamic** to reproduce results provided by the cfaOne, cfaHB, nnorOne, nnorHB, likertOne, likertHB2, catOne, and catHB from the **dynamic** package
progress	logical: if TRUE (default), progress bar will be displayed while fitting the CFA measurement model to the simulated samples. Note that a for loop is used when progress = TRUE, while the sapply function is used when progress = FALSE.
print	a character vector indicating the output shown on the console, i.e., "all" for all outputs, "summary" (default) for a summary of the specification in lavaan for the simulation, "model" for the lavaan model syntax for the CFA measurement model for each misspecification level specified for the simulation, "cutoff" (default) for the simulated fit index cutoffs.
digits	an integer value (default: 3) indicating the number of decimal places to be used for displaying fit indices.
plot	logical: if TRUE, distributions of fit indices for each level of misspecification is plotted.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file
width	a numeric value indicating the width argument (default: size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default: size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default: 600) in the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Value

Returns an object of class misty.object, which is a list with following entries:

`call`	function call
`type`	type of analysis
`args`	specification of function arguments
`model`	object or character string specified in the argument `model`
`data`	a data frame extracted from the object specified in the argument `model` or a data frame specified in the argument `data`
`sim.model`	a list of character strings indicating the lavaan model syntax for the CFA measurement model for each misspecification level specified for the simulation
`plot`	ggplot2 object when specifying `plot = TRUE`
`result`	list with results, i.e., `summary` for the summary of the specification in lavaan for the simulation, `summary.empirical` for the summary of the specification in lavaan for the fitted model, `fit.sim` for a list with data frames for the simulated fit indices, `fit.quant` for a list with data frames with the quantiles for the simulated fit indices, `fit.cutoff` for a data frame with the simulated fit index cutoffs and the specificity and sensitivity for each fit index, and `fit.emp` for the chi-square value and empirical fit indices of the fitted model.

Note

This function is based on the functions `cfaOne`, `cfaHB`, `nnorOne`, `nnorHB`, `likertOne`, `likertHB2`, `catOne`, and `catHB` from the **dynamic** package by Melissa Gordon Wolf and Daniel McNeish (2026).

Author(s)

Takuya Yanagida

References

Liu, X., & McNeish, D. (2025). Optimal number of replications for obtaining stable dynamic fit index cutoffs. *Educational and Psychological Measurement, 85*(3), 539–564. https://doi.org/10.1177/00131644241290172

McNeish, D. (2023). Dynamic fit index cutoffs for categorical factor analysis with Likert-type, ordinal, or binary responses. *American Psychologist, 78*(9), 1061–1075. https://doi.org/10.1037/amp0001213

McNeish, D. & Wolf, M. G. (2022). Dynamic fit cutoffs for one-factor models. *Behavior Research Methods, 55*, 1157-1174. https://doi.org/10.3758/s13428-022-01847-y

McNeish, D., & Wolf, M. G. (2023). Dynamic fit index cutoffs for confirmatory factor analysis models. *Psychological Methods, 28*(1), 61-88. https://doi.org/10.1037/met0000425

McNeish, D. (2024). Dynamic fit index cutoffs for treating likert items as continuous. *Psychological Methods*. Advance online publication. https://doi.org/10.1037/met0000683

Wolf, M. G., & McNeish, D. (2026). dynamic: DFI Cutoffs for Latent Variable Models. R package version 1.1.0. Retrieved from https://github.com/melissagwolf/dynamic

See Also

`item.cfa`

Examples

```
## Not run:
# Load lavaan package
library(lavaan)

#————————————————————————————————————————————————————————————————————————
# Object of Class misty

#————————————————————————————————————
## Multivariate Normality across all Items

# Conduct confirmatory factor analysis: Continuous items
mod1a.fit <- item.cfa(HolzingerSwineford1939, x1:x6, estimator = "ML")

# Example 1a: Simulate DFI cutoffs, multivariate normality
item.dfi(mod1a.fit, type = "norm")

#————————————————————————————————————
## Multivariate Non-Normality across all Items

# Conduct confirmatory factor analysis: Continuous items
mod1b.fit <- item.cfa(HolzingerSwineford1939, x1:x6)

# Example 1b: Simulate DFI cutoffs, multivariate non-normality (default)
item.dfi(mod1b.fit)

#————————————————————————————————————
## Likert-Type Items Treated as Continuous

# Conduct confirmatory factor analysis: Likert-type items as continuous
mod1c.fit <- item.cfa(data.items, pitem1, pitem2r, pitem3r, pitem4)

# Example 1c: Simulate DFI cutoffs, Likert-type
item.dfi(mod1c.fit, type = "likert")

#————————————————————————————————————
## Ordered-Categorical Items

# Conduct confirmatory factor analysis: Ordered-categorical items
mod1d.fit <- item.cfa(data.items, pitem1, pitem2r, pitem3r, pitem4, ordered = TRUE)

# Example 1d: Simulate DFI cutoffs, ordered-categorical
item.dfi(mod1d.fit)

#————————————————————————————————————————————————————————————————————————
# Object of Class lavaan

# Model specification
mod <- 'f =~ x1 + x2 + x3 + x4 + x5 + x6'

#————————————————————————————————————
## Multivariate Normality across all Items
```

```
# Model estimation
mod2a.fit <- cfa(mod, data = HolzingerSwineford1939, estimator = "ML")

# Example 2a: Simulate DFI cutoffs, multivariate normality
mod2a.dfi <- item.dfi(mod2a.fit, type = "norm")

#————————————————————————————————————
## Multivariate Non-Normality across all Items

# Model estimation
mod2b.fit <- cfa(mod, data = HolzingerSwineford1939, estimator = "MLR")

# Example 2b: Simulate DFI cutoffs, multivariate non-normality (default)
mod2b.fit <- item.dfi(mod2b.fit)

#————————————————————————————————————
## Arguments 'print' and 'level'

# Model estimation
mod2c.fit <- cfa(mod, data = HolzingerSwineford1939, estimator = "MLR")

# Example 2c: Simulate DFI cutoffs, print all outputs
mod2c.dfi <- item.dfi(mod2c.fit, print = "all")

# Example 2c: Print model syntax for each misspecification level
print(mod2c.dfi, print = "model")

# Example 2d: Print fit index cutoffs with 5 digits
print(mod2c.dfi, digits = 5)

# Example 2e: Simulate DFI cutoffs, simulate misspecification level 0 only
item.dfi(mod2c.fit, level = 0)

#————————————————————————————————————————————————————————————————————
# Character String

# Model specification
mod3 <- 'f =~ 0.42*x1 + 0.21*x2 + 0.20*x3 + 0.85*x4 + 0.85*x5 + 0.84*x6'

# Example 3a: Simulate DFI cutoffs, multivariate normality (default)
item.dfi(mod3, n = 301, estimator = "ML")

# Example 3b: Simulate DFI cutoffs, multivariate non-normality
item.dfi(mod3, n = 301, data = HolzingerSwineford1939, estimator = "MLR")

#————————————————————————————————————————————————————————————————————
# Plot

# Conduct confirmatory factor analysis
mod3.fit <- item.cfa(HolzingerSwineford1939, x1:x6)

# Example 4: Plot distributions of fit indices for each level of misspecification
```

```
item.dfi(mod3.fit, plot = TRUE)

#═══════════════════════════════════════════════════════════════════════
# Write Results and Save Plot

# Conduct confirmatory factor analysis
mod4.fit <- item.cfa(HolzingerSwineford1939, x1:x6)

# Example 4a: Write Results into a text file
item.dfi(mod4.fit, write = "CFA_DFI.txt")

# Example 4b: Write Results into an Excel file
item.dfi(mod4.fit, write = "CFA_DFI.xlsx")

# Example 4c: Save Plot of distributions of fit indices
item.dfi(mod4.fit, plot = TRUE, filename = "CFA_DFI.png", width = 10, height = 7)

## End(Not run)
```

`item.distract` *Distractor Analysis for Multiple-Choice Items*

Description

This function conducts distractor analysis for multiple-choice test items, i.e., computes attractor-distractor selection frequency, percentage, and total point-biserial or biserial correlation.

Usage

```
item.distract(data, ..., key, exclude = NULL, na.keep = FALSE, correct = TRUE,
              method = c("pbiser", "biser"), ml = FALSE,
              print = c("all", "nUQ", "key", "n", "perc", "r"),
              color = "default", style = c("regular", "bold", "italic"),
              na.omit = FALSE, digits = 1, r.digits = 3, as.na = NULL,
              write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame. Note that missing values should be represented with `NA` and not with a valid code. The argument `as.na` can be used to convert values to `NA` before conducting the analysis.
`...`	an expression indicating the variable names in `data` e.g., `item.distract(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`key`	a numeric or character vector of the answer key with a length matching the number of variables specified in the argument `data` or selected using the argument `....`
`exclude`	a character vector indicating items to be excluded from the analysis.

na.keep	logical: if FALSE (default), missing values are coded as 0 when scoring items (i.e., item not solved) and creating groups (i.e., answer option not selected).
correct	logical: if TRUE (default), the corrected distractor-total correlation is computed.
method	a character string indicating which correlation coefficient is used for the item-total correlation, i.e., "pbiser" for the point-biserial correlation coefficient when data are dichotomous, "biser" for the biserial correlation coefficient.
ml	logical: if FALSE (default), a two-step approximation is used to compute the biserial correlation coefficient (method = "biser"), while the maximum-likelihood (ML) estimate is computed if TRUE. Note that ML estimation is computationally expensive, i.e., takes a lot of time.
print	a character vector indicating which results to print, i.e., "all" for all results, "nUQ" for the number of unique elements after omitting missing values, "key" for the answer key, "n" for the distractor selection frequency, "perc" for the distractor selection percentage, and "r" for the (corrected) distractor-total correlation.
color	a character string indicating the text color for highlighting the attractor frequency, percentage, and attractor-total correlation, , i.e., "default" (default) for the default text color without color coding and various text colors for highlighting like "red", "b.red", "green", "b.green", "blue", or "b.blue", see the help page of the chr.color function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to "default" when using R Markdown and when writing the output into a text file (.txt).
style	a character vector indicating the font style, i.e., "regular" (default) for regular text, "bold" for bold text, and "italic" for italic text. Note that the font style "bold" and "italic" can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument color needs to be specified to change the style of the text, e.g. color = "black" and style = "bold" to for bold text.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
digits	an integer value indicating the number of decimal places to be used to display descriptive statistics, e.g., mean, median, and standard deviation.
r.digits	an integer value indicating the number of decimal places to be used to display the item difficulty, item-total correlation, and the coefficient alpha if the item is deleted.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame including all variables used in the analysis
`key`	list with the answer key and the position of the answer key for each item
`args`	specification of function arguments
`result`	result table

Note

This function uses a modified copy of the `polyserial()` function in the **polycor** package by John Fox when requesting distractor-total biserial correlation.

Author(s)

Takuya Yanagida

References

Fox, J. (2025). *polycor: Polychoric and polyserial correlations*. R package version 0.8-2. https://doi.org/10.32614/CRAN.package.polycor

See Also

`item.stats`, `item.alpha`, `item.omega`, `item.cfa`, `item.reverse`, `item.scores`

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Multiple-Choice Items

# Answer key
key <- c("D", "B", "C", "C", "B", "B", "B", "A", "A", "D")

# Example 1a: Corrected distractor-total point-biserial correlation
item.distract(data.items, +mitem, key = key)

# Example 1b: Corrected distractor-total biserial correlation
item.distract(data.items, +mitem, key = key, method = "biser")

# Example 1c: Keep missing values, i.e., do not convert NA into 0
item.distract(data.items, +mitem, key = key, na.keep = TRUE)

# Example 1d: Print corrected distractor-total point-biserial correlation only
item.distract(data.items, +mitem, key = key, print = "r")
```

```
# Example 1e: Highlight attractor frequency, percentage, and correlation in bright red
item.distract(data.items, +mitem, key = key, na.keep = TRUE, color = "b.red")

# Example 1f: Highlight attractor frequency, percentage, and correlation in boldface
item.distract(data.items, +mitem, key = key, na.keep = TRUE, color = "black",
              style = "bold")

#-----------------------------------------------------------------------
# Write Results

# Example 2a: Write Results into a text file
item.distract(data.items, +mitem, key = key, write = "Item-Distract.txt", output = FALSE)

# Example 2b: Write Results into an Excel file
item.distract(data.items, +mitem, key = key, write = "Item-Distracts.xlsx", output = FALSE)

## End(Not run)
```

`item.invar` *Between-Group and Longitudinal Measurement Invariance Evaluation*

Description

This function evaluates configural, (threshold), metric, scalar, and strict between-group or longitudinal (partial) measurement invariance using confirmatory factor analysis with continuous or ordered categorical indicators by calling the `cfa` function in the R package **lavaan**. Measurement invariance evaluation for measurement models with ordered categorical indicators utilizes the Wu and Estabrook (2016) approach to model identification and constraints to investigate measurement invariance. By default, the function evaluates configural, metric, and scalar measurement invariance for measurement models with continuous indicators, while the function evaluates configural, threshold, metric, scalar, and strict measurement invariance for measurement models with ordered categorical indicators given at least four response categories for each indicator by providing a table with model fit information (i.e., chi-square test, fit indices based on a proper null model, and information criteria) and model comparison (i.e., chi-square difference test, change in fit indices, and change in information criteria). Additionally, variance-covariance coverage of the data, descriptive statistics, parameter estimates, modification indices, residual correlation matrix, and relative Opdyke distribution percentile matrix can be requested by specifying the argument `print`.

Usage

```
item.invar(data, ..., model = NULL, group = NULL, cluster = NULL,
          long = FALSE, ordered = FALSE, parameterization = c("delta", "theta"),
           rescov = NULL, rescov.long = TRUE,
           invar = c("config", "thres", "metric", "scalar", "strict"),
           partial = NULL, ident = c("marker", "var", "effect"),
           estimator = c("ML", "MLM", "MLMV", "MLMVS", "MLF", "MLR",
                         "GLS", "WLS", "DWLS", "WLSM", "WLSMV", "WLSMVS",
```

```
                    "ULS", "ULSM", "ULSMV", "ULSMVS", "DLS", "PML"),
  missing = c("listwise", "pairwise", "fiml", "two.stage",
                  "robust.two.stage", "doubly.robust"), null.model = TRUE,
 print = c("all", "summary", "partial", "coverage", "descript", "fit",
                "est", "modind", "resid", "opdyke"),
print.fit = c("all", "standard", "scaled", "robust"), mod.minval = 6.63,
  resid.minval = 0.1, opdyke.prec = 1, opdyke.minmax = c(0.40, 0.60),
  color = "b.red", style = c("regular", "bold", "italic"),
 lavaan.run = TRUE, se = NULL, digits = 3, p.digits = 3, as.na = NULL,
  write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. If `model = NULL`, confirmatory factor analysis based on a measurement model with one factor labeled `f` comprising all variables in the data frame specified in `x` for evaluating between-group measurement invariance for the grouping variable specified in the argument `group` is conducted. Longitudinal measurement invariance evaluation can only be conducted by specifying the model using the argument `model`. Note that the cluster variable is excluded from `x` when specifying `cluster`. If `model` is specified, the data frame needs to contain all variables used in the argument `model` and the cluster variable when specifying the name of the cluster variable in the argument `cluster`.

`...` an expression indicating the variable names in `data`, e.g., `item.invar(dat, x1, x2, x2, group = "group")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`model` a character vector specifying a measurement model with one factor, or a list of character vectors for specifying a measurement model with more than one factor for evaluating between-group measurement invariance when `long = FALSE` or a list of character vectors for specifying a measurement model with one factor for each time of measurement for evaluating longitudinal measurement invariance when specifying `long = TRUE`. For example, `model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `f` comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"),factor2 = c("x5", "x6", "x7", "x8"))` for specifying a measurement model with two latent factors labeled `factor1` and `factor2` each comprising four indicators for evaluating between-group measurement invariance, or `model = list(time1 = c("ax1", "ax2", "ax3", "ax4"),time2 = c("bx1", "bx2", "bx3", "bx4"),time3 = c("cx1", "cx2", "cx3", "cx4"))` for specifying a longitudinal measurement model with three time points comprising four indicators at each time point. This function cannot evaluate longitudinal measurement invariance for a measurement model with more than one factor. Note that the name of each list element is used to label factors, i.e., all list elements need to be named, otherwise factors are labeled with `"f1", "f2", "f3"` when `long = FALSE` and with `"t1", "t2", "t3"` when `long = TRUE` and so on.

`group` either a character string indicating the variable name of the grouping variable in the data frame specified in `x` or a vector representing the groups for conducting multiple-group analysis to evaluate between-group measurement invariance.

`cluster` either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable) for computing scaled chi-square test statistic that takes into account non-independence of observations. Note that this option is not available when evaluating measurement invariance for ordered categorical indicators by specifying `ordered = TRUE)`.

`long` logical: if `TRUE`, longitudinal measurement invariance evaluation is conducted. The longitudinal measurement model is specified by using the argument `model`. Note that this function can only deal with a measurement model with one factor at each time point when investigating longitudinal measurement invariance. Moreover, this function can only evaluate either between-group or longitudinal measurement invariance, but not both at the same time.

`ordered` logical: if `TRUE`, all indicator variables of the measurement model are treated as ordered categorical variables, i.e., measurement invariance evaluation utilizes the Wu and Estabrook (2016) approach to model identification and constraints for investigating measurement invariance. Note that all indicators variables need to have the same number of response categories, either two (binary), three (ternary), or more than three response categories. Accordingly, zero cell counts are not allowed, e.g., zero observations for a response category of an indicator within a group when investigating between-group measurement invariance or zero observations for a response category of an indicator at a time point when investigating longitudinal measurement invariance.

`parameterization`
a character string only used when treating indicators of the measurement model as ordered categorical (`ordinal = TRUE`), i.e., `"delta"` (default) for delta parameterization or `"theta"` for theta parameterization.

`rescov` a character vector or a list of character vectors for specifying residual covariances, e.g., `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2`, or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.

`rescov.long` logical: if `TRUE` (default), residual covariances between parallel indicators are estimated across time when evaluating longitudinal measurement invariance (`long = TRUE`), i.e., residual variances of the same indicators that are measured at different time points are correlated across all possible time points. Note that residual covariances should be estimated even if the parameter estimates are statistically not significant since indicator-specific systematic variance is likely to correlate with itself over time (Little, 2013, p. 164).

`invar` a character string indicating the level of measurement invariance to be evaluated, i.e., `config` to evaluate configural measurement invariance (i.e., same factor structure across groups or time), `thres` to evaluate configural, and threshold measurement invariance (i.e., equal item-specific threshold parameters across group or time), `metric` to evaluate configural, threshold and metric measurement invariance (i.e., equal factor loadings across groups or time), `scalar` (default when `ordered = FALSE`) to evaluate configural, threshold, metric and scalar measurement invariance (i.e., equal intercepts across groups or time), and `strict` (default when `ordered = TRUE`) to evaluate configural, threshold, metric, scalar,

and strict measurement invariance (i.e., equal residual variances or scaling factors across groups or time). Note that threshold measurement invariance is only available when evaluating measurement invariance for ordered categorical indicators. In this case, threshold measurement invariance can only be investigated when all indicators have at least four response categories. In addition, metric measurement invariance cannot be investigated when all indicators have only two response categories, i.e., binary indicators.

`partial` a list of character vectors and/or list(s) named `thres` for freeing thresholds, `load` for freeing factor loadings, `inter` for freeing intercepts, and/or `resid` for freeing residual variances. More specifically, a list of named character vectors `load`, `inter`, and/or `resid` are needed when investigating partial between-group measurement invariance for continuous indicators based on two groups (see Example 4a) or longitudinal partial measurement invariance for continuous indicators based on two time points (see Example 10) or more than two time points However, a list of named lists labled `load`, `inter`, and/or `resid` are needed when investigating partial between-group measurement invariance for continuous indicators based on more than two groups (see Example 4b). A list including a list named `thres` of character vectors named a after the indicators for which threshold parameters (e.g., `"t1"` for threshold 1) are to be freed along with named character vectors `load`, `inter`, and/or `resid` are needed when investigating partial between-group measurement invariance for ordered categorical indicators based on two groups (see Example 12a) or longitudinal partial measurement invariance for ordered categorical indicators based on two time points (see Example 14) or more than two time points. However, a list including a list named `thres` of lists named after the indicators for which threshold parameters are freed (e.g., `item1`) of named character vectors specifying the threshold parameter along with lists named `load`, `inter`, and/or `resid` are needed when investigating partial between-group measurement invariance for ordered categorical indicators based on more than two groups (see Example 12b). Groups are denoted with `"g"` following the group number, e.g. `load = list(x4 = c("g1", "g3"))` for freeing factor loadings of indicator `x4` in group 1 and group 3. Note that at least two invariant indicators per latent variable are needed for a partial measurement invariance model. Otherwise there might be issues with model non-identification.

`ident` a character string indicating the method used for identifying and scaling latent variables, i.e., `"marker"` for the marker variable method fixing the first factor loading of the latent variable to 1 and fixing the first intercept to 0, `"var"` (default) for the fixed variance method fixing the variance of the latent variable to 1 and the latent mean to 0, or `"effect"` for the effects-coding method using equality constraints so that the average of the factor loading of the latent variable equals 1 and the sum of intercepts equals 0. Note that measurement invariance evaluation for ordered categorical indicators can only be conducted based on the fixed variance method (`"var"`).

`estimator` a character string indicating the estimator to be used (see 'Details' in the help page of the `item.cfa()` function). By default, `"MLR"` is used for CFA models with continuous indicators and `"WLSMV"` is used for CFA models with ordered categorical indicators. Note that the estimators `"ML"`, `"MLM"`, `"MLMV"`, `"MLMVS"`, `"MLF"` and `"MLR"` are not available when `ordered = TRUE`.

missing — a character string indicating how to deal with missing data, i.e., "listwise" for listwise deletion, "pairwise" for pairwise deletion, "fiml" for full information maximum likelihood method, "two.stage" for two-stage maximum likelihood method, "robust.two.stage" for robust two-stage maximum likelihood method, and "doubly-robust" for doubly-robust method (see 'Details' in the help page of theitem.cfa() function). By default, "fiml" is used for CFA models with continuous indicators and "listwise" is used for CFA models with ordered categorical indicators given that "fiml" is not available for a limited-information estimator used to estimate the CFA model with ordered categorical indicators. Note that the argument missing switches to listwise when the data set is complete. Also note that the robust CFI, TLI, and RMSEA are different in complete data depending on whether FIML or listwise deletion was specified when estimating the model in lavaan.

null.model — logical: if TRUE (default), the proper null model for computing incremental fit indices (i.e., CFI and TLI) is used, i.e., means and variances of the indicators are constrained to be equal across group or time in the null model (Little, 2013, p. 112). Note that the function does not provide the proper null model specification when evaluating measurement invariance for ordered categorical indicators i.e., the argument will switch to FALSE when specifying ordered = TRUE).

print — a character vector indicating which results to show on the console, i.e. "all" for all results, "summary" for a summary of the specification (e.g., estimation and optimization method, test statistic, missing data handling, and identification method), "partial" for a summary of the partial measurement invariance specification listing parameters that are freely estimated when partial is not NULL, "coverage" for the variance-covariance coverage of the data, "descript" for descriptive statistics for continuous variables (ordered = FALSE) and item frequencies for ordered categorical variable (ordered = TRUE), "fit" for model fit and model comparison, "est" for parameter estimates, "modind" for modification indices, "resid" for the residual correlation matrix and standardized residual means, and "opdyke" for the discrepancy between model-implied and observed correlation expressed in terms of the relative percentile of an Opdyke distribution (McNeish, 2025). By default, a summary of the specification, model fit, and parameter estimates are printed. Note that parameter estimates, modification indices, and residual correlation matrix is only provided for the model investigating the level of measurement invariance specified in the argument "invar".

print.fit — a character vector indicating which version of the CFI, TLI, and RMSEA to show on the console when using a robust estimation method involving a scaling correction factor, i.e., "all" for all versions of the CFI, TLI, and RMSEA, "standard" (default when estimator is one of "ML", "MLF", "GLS", "WLS", "DWLS", "ULS", "PML") for fit indices without any non-normality correction, "scaled" (default when ordered = TRUE) for population-corrected robust fit indices with ad hoc non-normality correction, and robust (default when estimator is one of "MLM", "MLMV", "MLMVS", "MLR", "WLSM", "WLSMV", "ULSM", "ULSMV", "DLS") for sample-corrected robust fit indices based on formula provided by Li and Bentler (2006) and Brosseau-Liard and Savalei (2014).

mod.minval — numeric value to filter modification indices and only show modifications with a modification index value equal or higher than this minimum value. By default,

modification indices equal or higher 6.63 are printed. Note that a modification index value of 6.63 is equivalent to a significance level of $\alpha = .01$.

`resid.minval` numeric value indicating the minimum absolute residual correlation coefficients and standardized means to highlight in boldface. By default, absolute residual correlation coefficients and standardized means equal or higher 0.1 are highlighted. Note that highlighting can be disabled by setting the minimum value to 1.

`opdyke.prec` a numeric value indicating the precision of the probability density function calculations of the Opdyke distribution. The default is `1` which calculates the PDF for polar angles between $(0, pi)$ in 0.01 increments. Specifying `10` calculates the PDF polar angles between $(0, pi)$ in 0.001 increments, which takes considerably longer, especially if there are many correlation elements.

`opdyke.minmax` a numeric vector with two elements indicating the minimum and maximum percentile of the Opdyke distribution that is considered to be acceptably close to the observed correlation represented by the Opdyke distribution median. Predicted correlation outside the range will be color highlighted in line with to the argument `color`.

`color` a character string indicating the text color for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval` and predicted correlations outside the minimum and maximum percentile of the Opdyke distribution, i.e., i.e., `"default"` for the default text color without color coding and various text colors for highlighting `"red"`, `"b.red"` (default), `"green"`, `"b.green"`, `"blue"`, or `"b.blue"`, see the help page of the `chr.color` function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to `"default"` when using R Markdown and when writing the output into a text file (`.txt`).

`style` a character vector indicating the font style for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval`, i.e., `"regular"` (default) for regular text, `"bold"` for bold text, and `"italic"` for italic text. Note that the font style `"bold"` and `"italic"` can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument `color` needs to be specified to change the style of the text, e.g. `color = "black"` and `style = "bold"` to for bold text.

`lavaan.run` logical: if `TRUE` (default), all models for evaluating measurement invariance will be estimated by using the `cfa()` function from the R package lavaan.

`se` internal argument only used in the `item.nonequi()` function, this argument should never be specified.

`digits` an integer value indicating the number of decimal places to be used for displaying results. Note that information criteria and chi-square test statistic are printed with `digits` minus 1 decimal places.

`p.digits` an integer value indicating the number of decimal places to be used for displaying *p*-values, covariance coverage (i.e., `p.digits - 1`), and residual correlation coefficients.

`as.na` a numeric vector indicating user-defined missing values, i.e., these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x` but not to `group` or `cluster`.

write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked and convergence and model identification checks are conducted for all estimated models.
output	logical: if TRUE (default), output is shown.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame including all variables used in the analysis, i.e., indicators for the factor, grouping variable and cluster variable
args	specification of function arguments
model	list with specified model for the for the configural (config), threshold (thresh), metric (metric), scalar (scalar), and strict invariance model (strict)
model.fit	list with fitted lavaan object of the configural, metric, scalar, and strict invariance model
check	list with the results of the convergence and model identification check for the configural (config), threshold (thresh), metric (metric), scalar (scalar), and strict invariance model (strict)
result	list with result tables, i.e., summary for the summary of the specification, e.g., estimation method or missing data handling in lavaan, partial for the summary of the partial invariance specification, coverage for the variance-covariance coverage of the data, descript list with descriptive statistics (stat) and frequencies (freq), fit for a list with model fit based on standard, scaled, and robust fit indices, param for a list with parameter estimates for the configural, metric, scalar, and strict invariance model, modind for the list with modification indices for the configural, metric, scalar, and strict invariance model, score for the list with result of the score tests for constrained parameters for the threshold, metric, scalar, and strict invariance model, resid for the list with residual correlation matrices and standardized residual means for the configural, threshold, metric, scalar, and strict invariance model, and opdyke for the list with relative Opdyke distribution percentile matrix for the configural, threshold, metric, scalar, and strict invariance model

Note

The function uses the functions cfa, fitmeasures ,lavInspect, lavTech, lavTestLRT, lavTestScore, modindices, parameterEstimates, parTable, and standardizedsolution provided in the R package **lavaan** by Yves Rosseel (2012). The relative Opdyke distribution percentile matrix is based on the functions opdyke and opdyke.percentiles provided in the R package **opddyke** by Dan McNeish (2026).

Author(s)

Takuya Yanagida

References

Brosseau-Liard, P. E., & Savalei, V. (2014) Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research, 49*, 460-470. https://doi.org/10.1080/00273171.2014.933697

Li, L., & Bentler, P. M. (2006). Robust statistical tests for evaluating the hypothesis of close fit of misspecified mean and covariance structural models. *UCLA Statistics Preprint #506*. University of California.

Little, T. D. (2013). *Longitudinal structural equation modeling*. Guilford Press.

McNeish D (2026). opdyke: Opdyke Percentiles for Approximate Local Fit in Structural Equation Models. R package version 1.0.0. Retrieved from https://github.com/dmcneish18/opdyke

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

Wu, H., & Estabrook, R. (2016). Identification of confirmatory factor analysis models of different levels of invariance for ordered categorical outcomes. *Psychometrika, 81*(4), 1014–1045. doi:10.1007/s11336-016-9506-0

See Also

item.noninvar, item.cfa, multilevel.invar

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Between-Group Measurement Invariance: Continuous Indicators

#————————————————————————————————————
# Measurement Model with One Factor

# Example 1a: Model specification using the argument '...'
item.invar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school")

# Example 1b: Alternative model specification without using the argument '...'
item.invar(HolzingerSwineford1939[, c("x1", "x2", "x3", "x4")],
           group = HolzingerSwineford1939$sex)

# Example 1c: Alternative model specification without using the argument '...'
item.invar(HolzingerSwineford1939[, c("x1", "x2", "x3", "x4", "school")], group = "school")

# Example 1d: Alternative model specification using the argument 'model'
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"), group = "school")
```

```
#————————————————————————————————————
# Measurement Model with Two Factors

# Example 2: Model specification using the argument 'model'
item.invar(HolzingerSwineford1939,
           model = list(c("x1", "x2", "x3", "x4"), c("x5", "x6", "x7", "x8")),
           group = "school")

#————————————————————————————————————
# Configural, Metric, Scalar, and Strict Measurement Invariance

# Example 3: Evaluate configural, metric, scalar, and strict measurement invariance
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", invar = "strict")

#————————————————————————————————————
# Between-Group Partial Measurement Invariance

# Example 4a: Two Groups
#             Free factor loadings for 'x2' and 'x3'
#             Free intercept for 'x1'
#             Free residual variance for 'x4'
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", invar = "strict",
           partial = list(load = c("x2", "x3"),
                          inter = "x1",
                          resid = "x4"))

# Example 4b: More than Two Groups
#             Free factor loading for 'x2' in group 2
#             Free factor loading for 'x4' in group 1 and 3
#             Free intercept for 'x1' in group 3
#             Free residual variance for 'x3' in group 1 and 3
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "ageyr", invar = "strict",
           partial = list(load = list(x2 = "g2", x4 = c("g1", "g3")),
                          inter = list(x1 = "g3"),
                          resid = list(x3 = c("g1", "g3"))))

#————————————————————————————————————
# Residual Covariances

# Example 5a: One residual covariance
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           rescov = c("x3", "x4"), group = "school")

# Example 5b: Two residual covariances
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           rescov = list(c("x1", "x4"), c("x3", "x4")), group = "school")

#————————————————————————————————————
# Default Null Model
```

```
# Example 6: Specify default null model for computing incremental fit indices
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", null.model = FALSE)

#————————————————————————————————————
# Arguments 'print', 'print.fit', 'mod.minval', 'resid.minval', and 'opdyke.minmax'

# Example 7a: Request all results
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print = "all")

# Example 7b: Request fit indices with ad hoc non-normality correction
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print.fit = "scaled")

# Example 7c: Request modification indices with value equal or higher than 2
# highlight absolute residual correlations equal or higher than 0.05
# highlight Opdyke distribution percentiles outside 0.45 and 0.55
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print = c("modind", "resid", "opdyke"),
           mod.minval = 2, resid.minval = 0.05, opdyke.minmax = c(0.45, 0.55))

#————————————————————————————————————
# Model Syntax and lavaan Summary of the Estimated Model

# Example 8a: Model specification using the argument '...'
mod1 <- item.invar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school",
                   output = FALSE)

# lavaan summary of the scalar invariance model
lavaan::summary(mod1$model.fit$scalar, standardized = TRUE, fit.measures = TRUE)

# Example 8b: Do not estimate any models
mod2 <- item.invar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school",
                   lavaan.run = FALSE)

# lavaan model syntax metric invariance model
cat(mod2$model$metric)

# lavaan model syntax scalar invariance model
cat(mod2$model$scalar)

#————————————————————————————————————————————————————————————————————
# Longitudinal Measurement Invariance: Continuous Indicators

# Example 9: Two time points with three indicators at each time point
item.invar(HolzingerSwineford1939,
           model = list(c("x1", "x2", "x3"), c("x5", "x6", "x7")), long = TRUE)

#————————————————————————————————————
# Longitudinal Partial Measurement Invariance

# Example 10: Two Time Points with three indicators at each time point
```

```
#             Free factor loading for 'x2'
#             Free intercepts for 'x1' and x2
item.invar(HolzingerSwineford1939,
           model = list(c("x1", "x2", "x3"), c("x5", "x6", "x7")), long = TRUE,
           partial = list(load = "x2",
                          inter = c("x1", "x2")))

#————————————————————————————————————————————————————————————————————————
# Between-Group Measurement Invariance: Ordered Categorical Indicators

# Example 11a: Delta parameterization (default)
item.invar(data.items, pitem1, pitem2r, pitem3r, pitem4, group = "group2",
           ordered = TRUE)

# Example 11b: Theta parameterization
item.invar(data.items, pitem1, pitem2r, pitem3r, pitem4, group = "group2", ordered = TRUE,
           parameterization = "theta")

#————————————————————————————————————
# Between-Group Partial Measurement Invariance

# Example 12a: Two Groups
#              Free 2nd and 3rd threshold of 'pitem1'
#              Free 1st threshold of 'pitem3r'
#              Free factor loadings for 'pitem2r' and 'pitem4'
#              Free intercept for 'pitem1'
#              Free residual variance for 'pitem3r'
item.invar(data.items, pitem1, pitem2r, pitem3r, pitem4, group = "group2",
           ordered = TRUE,
           partial = list(thres = list(pitem1 = c("t2", "t3"),
                                       pitem3r = "t1"),
                          load = c("pitem2r", "pitem4"),
                          inter = "pitem1",
                          resid = "pitem3r"))

# Example 12b: More than Two Groups
#              Free 1st threshold of 'pitem1' in group 1 and 2
#              Free 2nd threshold of 'pitem3r' in group 3
#              Free factor loadings for 'pitem2r' in group 1
#              Free intercept for 'pitem2r' in group 1
#              Free intercept for 'item3r' in group 2 and 4
#              Free residual variance for 'pitem1' in group 1 and 3
item.invar(data.items, pitem1, pitem2r, pitem3r, pitem4, group = "group4", ordered = TRUE,
           partial = list(thres = list(pitem1 = list(t1 = c("g1", "g2")),
                                       pitem3r = list(t2 = "g3")),
                          load  = list(pitem2r = "g1"),
                          inter = list(pitem2r = "g1", pitem3r = c("g2", "g4")),
                          resid = list(pitem1 = c("g1", "g3"))))

#————————————————————————————————————————————————————————————————————————
# Longitudinal Measurement Invariance: Ordered Categorical Indicators

# Example 13: Two Time Points
```

```
item.invar(data.items, model = list(c("pitem1", "pitem2r", "pitem3r"),
                                    c("pitem4", "pitem5", "pitem6")),
           long = TRUE, ordered = TRUE, invar = "thres")

#————————————————————————————————————
# Longitudinal Partial Measurement Invariance

# Example 14: Two Time Points
#             Free 2nd and 3rd threshold of 'pitem1'
#             Free 1st threshold of 'pitem3r'
#             Free factor loading of 'pitem1'
item.invar(data.items, model = list(c("pitem1", "pitem2r", "pitem3r"),
                                    c("pitem4", "pitem5", "pitem6")),
           long = TRUE, ordered = TRUE, invar = "thres",
           partial = list(thres = list(pitem1 = c("t2", "t3"), pitem3r = "t1"),
                          load  = "pitem1"))

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 15a: Write Results into a text file
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print = "all", write = "Invariance.txt", output = FALSE)

# Example 15b: Write Results into an Excel file
item.invar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print = "all", write = "Invariance.xlsx", output = FALSE)

## End(Not run)
```

---

item.noninvar *Effect Size Measure of Measurement Non-Invariance, dMACS*

Description

This function computes the effect size measure dMACS by Nye and Drasgow (2011) and the signed dMACS by Nye et al. (2019) for evaluating the magnitude and the direction of between-group and longitudinal measurement non-invariance or non-equivalence for continuous and ordered categorical items and also computes the expected bias in the mean and variance of the total score.

Usage

```
item.noninvar(data = NULL, ..., object = NULL, model = NULL, group = NULL,
              ref = NULL, pooled = TRUE, signed = FALSE, cluster = NULL,
              long = FALSE, ordered = FALSE, rescov = NULL, rescov.long = TRUE,
              ident = c("marker", "var", "effect"),
              estimator = c("ML", "MLM", "MLMV", "MLMVS", "MLF", "MLR",
                            "GLS", "WLS", "DWLS", "WLSM", "WLSMV", "WLSMVS",
                            "ULS", "ULSM", "ULSMV", "ULSMVS", "DLS", "PML"),
```

```
missing = c("listwise", "pairwise", "fiml", "two.stage",
            "robust.two.stage", "doubly.robust"),
print = c("all", "summary", "dmacs", "bias"),
digits = 3, as.na = NULL, write = NULL, append = TRUE,
check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. If `model = NULL`, confirmatory factor analysis based on a measurement model with one factor labeled `f` comprising all variables in the data frame specified in `data` for evaluating between-group measurement non-invariance for the grouping variable specified in the argument `group` is conducted. Longitudinal measurement non-invariance evaluation can only be conducted by specifying the model using the argument `model`. Note that the cluster variable is excluded from `data` when specifying `cluster`. If `model` is specified, the data frame needs to contain all variables used in the argument `model` and the cluster variable when specifying the name of the cluster variable in the argument `cluster`.

`...` an expression indicating the variable names in `data`, e.g., `item.noninvar(dat, x1, x2, x2, group = "group")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`object` an object of class lavaan, i.e., a fitted latent variable model. Between-group measurement non-invariance is evaluated when specifying a fitted multiple-group model, while longitudinal measurement non-invariance is evaluated when specifying a fitted single-group model with at least two latent variables each representing a factor at different time points.

`model` a character vector specifying a measurement model with one factor, or a list of character vectors for specifying a measurement model with more than one factor for evaluating between-group measurement non-invariance when `long = FALSE` or a list of character vectors for specifying a measurement model with one factor for each time of measurement for evaluating longitudinal measurement non-invariance when specifying `long = TRUE`. For example, `model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `f` comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"),factor2 = c("x5", "x6", "x7", "x8"))` for specifying a measurement model with two latent factors labeled `factor1` and `factor2` each comprising four indicators for evaluating between-group measurement non-invariance, or `model = list(time1 = c("ax1", "ax2", "ax3", "ax4"),time2 = c("bx1", "bx2", "bx3", "bx4"),time3 = c("cx1", "cx2", "cx3", "cx4"))` for specifying a longitudinal measurement model with three time points comprising four indicators at each time point. This function cannot evaluate longitudinal measurement invariance for a measurement model with more than one factor. Note that the name of each list element is used to label factors, i.e., all list elements need to be named, otherwise factors are labeled with `"f1", "f2", "f3"` when `long = FALSE` and with `"t1", "t2", "t3"` when `long = TRUE` and so on.

`group` either a character string indicating the variable name of the grouping variable in the data frame specified in `data` or a vector representing the groups for conducting multiple-group analysis to evaluate between-group measurement non-invariance. Note that when specifying a grouping variable with more than two

groups, the function compares each group with the reference group specified in the argument `ref`.

`ref` a numeric value or character string indicating the name of the reference group or reference time point. By default, the the first group or time point is used as reference.

`pooled` logical: if `TRUE` (default), the item-level pooled standard deviation is used in the denominator of the effect size measure, if `FALSE` the item-level standard deviation of the reference group or reference time point is used.

`signed` logical: if `TRUE`, the signed dMACS is computed that incorporates the unsquared differences between groups or time points illustrating the direction of the differences and allowing effects in opposite directions to cancel out (see Nye et al., 2019).

`cluster` either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable). Note that this option is not available when evaluating measurement invariance for ordered categorical indicators by specifying `ordered = TRUE)`.

`long` logical: if `TRUE`, longitudinal measurement non-invariance evaluation is conducted. The longitudinal measurement model is specified by using the argument `model`. Note that this function can only deal with a measurement model with one factor at each time point when investigating longitudinal measurement non-invariance. Moreover, this function can only evaluate either between-group or longitudinal measurement non-invariance, but not both at the same time.

`ordered` logical: if `TRUE`, all indicator variables of the measurement model are treated as ordered categorical variables. Note that the function only supports delta parameterization. Also note that all indicators variables need to have the same number of response categories. Accordingly, zero cell counts are not allowed, e.g., zero observations for a response category of an indicator within a group when investigating between-group measurement non-invariance or zero observations for a response category of an indicator at a time point when investigating longitudinal measurement non-invariance.

`rescov` a character vector or a list of character vectors for specifying residual covariances, e.g., `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2`, or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.

`rescov.long` logical: if `TRUE` (default), residual covariances between parallel indicators are estimated across time when evaluating longitudinal measurement non-invariance (`long = TRUE`), i.e., residual variances of the same indicators that are measured at different time points are correlated across all possible time points. Note that residual covariances should be estimated even if the parameter estimates are statistically not significant since indicator-specific systematic variance is likely to correlate with itself over time (Little, 2013, p. 164).

`ident` a character string indicating the method used for identifying and scaling latent variables, i.e., `"marker"` for the marker variable method fixing the first factor loading of the latent variable to 1 and fixing the first intercept to 0, `"var"` (default) for the fixed variance method fixing the variance of the latent variable to

1 and the latent mean to 0, or "effect" for the effects-coding method using equality constraints so that the average of the factor loading of the latent variable equals 1 and the sum of intercepts equals 0. Note that measurement non-invariance evaluation for ordered categorical indicators can only be conducted based on the fixed variance method ("var").

estimator — a character string indicating the estimator to be used (see 'Details' in the help page of the item.cfa() function). By default, "MLR" is used for CFA models with continuous indicators and "WLSMV" is used for CFA models with ordered categorical indicators. Note that the estimators "ML", "MLM", "MLMV", "MLMVS", "MLF" and "MLR" are not available when ordered = TRUE.

missing — a character string indicating how to deal with missing data, i.e., "listwise" for listwise deletion, "pairwise" for pairwise deletion, "fiml" for full information maximum likelihood method, "two.stage" for two-stage maximum likelihood method, "robust.two.stage" for robust two-stage maximum likelihood method, and "doubly-robust" for doubly-robust method (see 'Details' in the help page of theitem.cfa() function). By default, "fiml" is used for CFA models with continuous indicators and "listwise" is used for CFA models with ordered categorical indicators given that "fiml" is not available for a limited-information estimator used to estimate the CFA model with ordered categorical indicators.

print — a character vector indicating which results to show on the console, i.e. "all" for all results, "summary" for a summary of the specification, "dmacs" for the effect sizes measure dMACS, and "bias" for the expected bias in the mean and variance of the total score. By default, a summary of the specification and the effect size measure dMACS are printed.

digits — an integer value indicating the number of decimal places to be used for displaying results.

as.na — a numeric vector indicating user-defined missing values, i.e., these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data but not to group or cluster.

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification is checked and convergence and model identification checks are conducted for all estimated models.

output — logical: if TRUE (default), output is shown.

Details

Nye and Drasgow (2011) introduced the effect size measure d_{MACS} (Mean and Covariance Structure) for evaluating measurement non-invariance at the item level on a standardized metric similar to Cohen's *d* (1988) or Glass's (1976) measures:

Effect Size Measure d_{MACS} d_{MACS} (Nye & Drasgow, 2011) ranging $[0, \infty]$ is based on the predicted response $\hat{X}_{iR}$ to an item i for an individual in the reference group (or reference time point) R and the corresponding response $\hat{X}_{iF}$ for an individual in the focal group (or focal time point) F:

$$\hat{X}_{iR} = \tau_{iR} + \lambda_{iR}\xi$$
$$\hat{X}_{iF} = \tau_{iF} + \lambda_{iF}\xi$$

where τ_{iR} and τ_{iF} are the intercepts, λ_{iR} and λ_{iF} are the factor loadings of item i in the reference and focal group, and ξ is the score on the latent variable.

The effect size evaluating the magnitude of measurement non-invariance is a weighted average difference in predicted responses in standardized metric defined as:

$$d_{MACS} = \frac{1}{SD_{iP}} \sqrt{\int (\hat{X}_{iR} - \hat{X}_{iF}|\xi)^2 f_F(\xi) d\xi}$$

where SD_{iP} is the pooled within-group standard deviation of item i across reference and focal group given by

$$SD_{iP} = \frac{(N_R - 1)SD_R + (N_F - 1)SD_F}{(N_R - 1) + (N_F - 1)}$$

Note that $f_F(\xi)$ is the distribution of the latent trait ξ in the focal group, which is assumed to have a normal distribution with a mean and variance estimated from the latent factor in the focal group.

Effect Size Measure d_{MACS_Signed} d_{MACS_Signed} (Nye et al., 2019) ranging $[-\infty, \infty]$ incorporates the unsquared differences between predicted response to an item i between two groups:

$$d_{MACS\_Signed} = \frac{1}{SD_{iP}} \int (\hat{X}_{iR} - \hat{X}_{iF}|\xi) f_F(\xi) d\xi$$

Note that d_{MACS_Signed} provides complementary information to the unsigned version by (1) capturing the direction of the difference and (2) allowing cancellation of effects in opposite direction.

Guidelines for Interpreting Effect Size Measure d_{MACS} **and** d_{MACS_Signed} The effect size measures d_{MACS} and d_{MACS_Signed} represent the differences in both the factor loadings and the intercepts across two groups and can be interpreted based on following guidelines (see Nye et al., 2019):

- *Effect Size Measure* d_{MACS}:
  - Results of a simulation study provided benchmarks for interpreting d_{MACS}:
    - Small effect: 0.20
    - Medium effect: 0.40
    - Large effect: 0.70
  - The simulation study operationalized effect sizes empirically based on a literature review of journals in organizational behavior and entrepreneurship:
    - Difference in standardized factor loadings: 0.10 (small), 0.20 (medium), and 0.30 (large)

    * Difference in intercept: 0.25 (small), 0.50 (medium), and 0.75 (large)
  - Results also showed that when the sample size ($n_g = 250$) and/or the number of items ($k = 8$) were small, d_{MACS} can become greater than 0.20 due to poorly estimated model parameters.

  Note that d_{MACS} does not provide information about the direction of the effect, i.e., it is unclear which group the item is biased against.

- *Effect Size Measure* d_{MACS_Signed}:
  - Results of a simulation study provided benchmarks for interpreting d_{MACS_Signed}:
    * Small effect: $|0.40|$
    * Medium effect: $|0.60|$
    * Large effect: $|0.80|$
  - Simulation study investigated the practical importance of non-invariance when no true latent mean difference exist between groups, i.e., false positive results due to non-invariance:
    * d_{MACS_Signed} = |0.40| in one out of eight items results in Cohen's d ~ 0.13 and a .13 probability for finding statistically significant differences due to non-invariance.
    * d_{MACS_Signed} = |0.60| in one out of eight items results in Cohen's d ~ 0.26 and a .85 probability for finding statistically significant differences due to non-invariance.
    * d_{MACS_Signed} = |0.80| in one out of eight items results in Cohen's d ~ 0.26 and a 1.00 probability for finding statistically significant differences due to non-invariance.

Practical Consequences of Measurement Non-Invariance The practical consequences of non-invariance can be investigated by computing the amount of the observed mean and variance difference of a scale between groups that can be attributed to non-invariance:

$$\Delta mean(x_s) = \sum_{1}^{n} \int (\hat{X}_{iR} - \hat{X}_{iF}|\xi) f_F(\xi) d\xi$$

$$\Delta var(x_s) = 2C_i \lambda_{iR} \phi_F + C_i^2 \phi_F$$

where X_s is the scale score, λ_{iR} is the factor loading of item i in the reference group, C_i is the difference between the factor loading for item i in the reference and focal groups, and ϕ_F is the variance of the latent factor in the focal group. According to these formula, two items with high d_{MACS_Signed} in opposite directions can have no impact on $\Delta mean(x_s)$ and $\Delta var(x_s)$ due to the cancellation effect.

Note that with fewer items in the scale, the practical importance of a single item with high d_{MACS_Signed} would increase, while a single non-invariant item in a longer measure would have less practical importance. For example, the practical importance of a single non-invariant item with a $d_{MACS\_Signed} = 0.40$ in a 30-item measure would correspond to a Cohen's d value of 0.05 and a .14 probability of a statistically significant mean differences in the absence of a true differences between groups. That is, the same cutoffs used for an 8-item measure might not apply to a 30-item measure. Moreover, a measure with more than one non-invariant item would have a greater chance of distorting research outcomes.

In summary, the findings in Nye et al. (2019) suggest that a more nuanced interpretation of the effect size of non-invariance may be required. Accordingly, Lai et al. (2025) notet that cutoff

values should be used with caution as the interpretation of the magnitude of non-invariance should be based on many other factors, such as the construct of interest, the grouping variables, the main usage of the instrument, the context of the measurement, and so on.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame including all variables used in the analysis, i.e., indicators for the factor, grouping variable and cluster variable
`args`	specification of function arguments
`model`	model specification for the for the configural invariance model
`model.fit`	fitted lavaan object of the configural invariance model
`check`	list with the results of the convergence and model identification check for the configural invariance model
`result`	list with result tables, i.e., `summary` for the summary of the specification and `noninvar` for the dMACS effect size measure, expected bias in the mean total score, and expected bias in the variance of the total score

Note

This function is based on modified copies of the functions `dmacs_summary` `dmacs_summary_single`, `item_dmacs`, `expected_value`, `delta_mean_item` and `delta_var` from the **dmacs** package by David Dueber.

Author(s)

Takuya Yanagida

References

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Lawrence Erlbaum.

Dueber D (2026). *dmacs: Measurement Nonequivalence Effect Size Calculator*. R package version 0.1.0.9002. https://github.com/ddueber/dmacs

Glass, G. V. (1976). Primary, secondary, and meta-analysis of research. *Educational Researcher, 5*, 3-8.

Lai, M. H. C., Zhang, Y., Ozcan, M., Tse, W. W. Y., & Miles, A. (2025). fMACS: Generalizing dMACS effect size for measurement noninvariance with multiple groups and multiple grouping variables. *Structural Equation Modeling: A Multidisciplinary Journal, 32*(4), 638-646. https://doi.org/10.1080/10705511.2025.2484812

Nye, C. D., Bradburn, J., Olenick, J., Bialko, C., & Drasgow, F. (2019). How big are my effects? Examining the magnitude of the effect sizes in studies of measurement equivalence. *Organizational Research Methods, 22*(3), 678–709. https://doi.org/10.1177/1094428118761122

Nye, C., & Drasgow, F. (2011). Effect size indices for analyses of measurement equivalence: Understanding the practical importance of differences between groups. *Journal of Applied Psychology, 96*(5), 966-980.

See Also

item.invar, item.cfa, multilevel.invar

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#=====================================================================
# Between-Group Measurement Non-Invariance: Continuous Indicators

#=====================================
## Measurement Model with One Factor

# Example 1a: Model specification using the argument '...'
item.noninvar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school")

# Example 1b: Alternative model specification without using the argument '...'
item.noninvar(HolzingerSwineford1939[, c("x1", "x2", "x3", "x4")],
              group = HolzingerSwineford1939$school)

# Example 1c: Alternative model specification without using the argument '...'
item.noninvar(HolzingerSwineford1939[, c("x1", "x2", "x3", "x4", "school")], group = "school")

# Example 1d: Alternative model specification using the argument 'model'
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"), group = "school")

# Example 1e: Estimate model and specify the 'object' argument
fit <- cfa('f =~ x1 + x2 + x3 + x4', data = HolzingerSwineford1939, group = "school", std.lv = TRUE)
item.noninvar(object = fit)

#=====================================
## Measurement Model with Two Factors

# Example 2a: Model specification using the argument 'model'
item.noninvar(HolzingerSwineford1939,
              model = list(c("x1", "x2", "x3", "x4"), c("x5", "x6", "x7", "x8")),
              group = "school")

# Example 2b: Model specification using the argument 'model'
model <- 'f1 =~ x1 + x2 + x3 + x4
          f2 =~ x5 + x6 + x7 + x8'

fit <- cfa(model, data = HolzingerSwineford1939, group = "school", std.lv = TRUE)
item.noninvar(object = fit)
```

```
#————————————————————————————————————
## Signed dMACS and Reference Group

# Example 3a: Signed dMACS
item.noninvar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school", signed = TRUE)

# Example 3b: Specify reference group and use SD of the reference group
item.noninvar(HolzingerSwineford1939, x1, x2, x3, x4, group = "school",
              ref = "Pasteur", pooled = FALSE)

#————————————————————————————————————
## Residual Covariances

# Example 4a: One residual covariance
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
              rescov = c("x3", "x4"), group = "school")

# Example 4b: Two residual covariances
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
              rescov = list(c("x1", "x4"), c("x3", "x4")), group = "school")

#————————————————————————————————————
# Print Argument

# Example 5: Request all results
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
              group = "school", print = "all")

#————————————————————————————————————————————————————————————————————
# Longitudinal Measurement Non-Invariance: Continuous Indicators

# Example 6: Two time points with three indicators at each time point
item.noninvar(HolzingerSwineford1939,
              model = list(c("x1", "x2", "x3"), c("x5", "x6", "x7")), long = TRUE)

#————————————————————————————————————————————————————————————————————
# Between-Group Measurement Non-Invariance: Ordered Categorical Indicators

# Example 7: Two groups
item.noninvar(data.items, pitem1, pitem2r, pitem3r, pitem4, group = "group2",
              ordered = TRUE)

#————————————————————————————————————————————————————————————————————
# Longitudinal Measurement Non-Invariance: Ordered Categorical Indicators

# Example 8: Two Time Points
item.noninvar(data.items, model = list(c("pitem1", "pitem5", "pitem3r"),
                                       c("pitem4", "pitem2r", "pitem6")),
              long = TRUE, ordered = TRUE)

#————————————————————————————————————————————————————————————————————
# Write Results
```

```
# Example 9a: Write Results into a text file
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
            group = "school", print = "all", write = "Non-Invariance.txt", output = FALSE)

# Example 9b: Write Results into an Excel file
item.noninvar(HolzingerSwineford1939, model = c("x1", "x2", "x3", "x4"),
           group = "school", print = "all", write = "Non-Invariance.xlsx", output = FALSE)

## End(Not run)
```

---

item.omega *Coefficient Omega, Hierarchical Omega, and Categorical Omega*

Description

This function computes point estimate and confidence interval for the coefficient omega (McDonald, 1999), hierarchical coefficient omega (Kelley & Pornprasertmanit, 2016), and categorical coefficient omega (Green & Yang, 2009) along with standardized factor loadings and omega if item deleted. By default, the function computes coefficient omega based on maximum likelihood parameter (ML) estimates using full information maximum likelihood (FIML) method in the presence of missing data.

Usage

```
item.omega(data, ..., rescov = NULL, type = c("omega", "hierarch", "categ"),
           exclude = NULL, std = FALSE,
           estimator = c("ML", "GLS", "WLS", "DWLS", "ULS", "PML"),
           missing = c("listwise", "pairwise", "fiml"),
           print = c("all", "omega", "item"), digits = 2, r.digits = 3,
           conf.level = 0.95, as.na = NULL, write = NULL, append = TRUE,
           check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame. Note that at least three items are needed for computing coefficient omega
`...`	an expression indicating the variable names in `data` e.g., `item.omega(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`rescov`	a character vector or a list of character vectors for specifying residual covariances when computing coefficient omega, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2` or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.
`type`	a character string indicating the type of omega to be computed, i.e., `omega` (default) for coefficient omega, `hierarch` for hierarchical coefficient omega, and `categ` for categorical coefficient omega.

exclude	a character vector indicating items to be excluded from the analysis.
std	logical: if TRUE, the standardized coefficient omega is computed.
estimator	a character string indicating the estimator to be used (see 'Details' in the item.cfa function). By default, "ML" is used for computing (hierarchical) coefficient omega and "DWLS" is used for computing ordinal coefficient omega.
missing	a character string indicating how to deal with missing data. (see 'Details' in the item.cfa function). By default, full information maximum likelihood method (missing = "fiml") is used for computing (hierarchical) coefficient omega and pairwise deletion (missing = "pairwise") is used to compute coefficient omega.
print	a character vector indicating which results to show, i.e. "all" for all results "omega" (default) for the coefficient omega, and "item" for item statistics.
digits	an integer value indicating the number of decimal places to be used for displaying mean, standard deviation, minimum, and maximum.
r.digits	an integer value indicating the number of decimal places to be used for displaying alpha and standardized factor loadings.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Details

Coefficient Omega This function computes the coefficient omega by conducting a confirmatory factor analysis based on the congeneric measurement model (Graham, 2006) using the cfa() function in the **lavaan** package by Yves Rosseel (2019).

Confidence Interval The confidence interval for the (ordinal) coefficient alpha is computed using the procedure by Feldt et al. (1987). Note that there are at least 10 other procedures for computing the confidence interval (see Kelley and Pornprasertmanit, 2016), which are implemented in the ci.reliability() function in the **MBESSS** package by Ken Kelley (2019).

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame used for the current analysis

`args`	specification of function arguments
`model.fit`	fitted lavaan object (`mod.fit`)
`result`	list with result tables, i.e., `omega` for a table with coefficient omega and `itemstat` for a table with item statistics

Note

Computation of the hierarchical and categorical omega is based on the `ci.reliability()` function in the **MBESS** package by Ken Kelley (2019).

Author(s)

Takuya Yanagida

References

Feldt, L. S., Woodruff, D. J., & Salih, F. A. (1987). Statistical inference for coefficient alpha. *Applied Psychological Measurement*, 11 93-103. https://doi.org/10.1177/014662168701100107

Graham, J. M. (2006). Congeneric and (essentially) tau-equivalent estimates of score reliability: What they are and how to use them. *Educational and Psychological Measurement, 66*(6), 930–944. https://doi.org/10.1177/0013164406288165

Kelley, K., & Pornprasertmanit, S. (2016). Confidence intervals for population reliability coefficients: Evaluation of methods, recommendations, and software for composite measures. *Psychological Methods, 21*, 69-92. https://doi.org/10.1037/a0040086.

Ken Kelley (2019). *MBESS: The MBESS R Package*. R package version 4.6.0. https://CRAN.R-project.org/package=MBESS

McDonald, R. P. (1999). *Test theory: A unified treatment*. Lawrence Erlbaum Associates Publishers.

See Also

`item.omega`, `item.cfa`, `item.invar`, `item.reverse`, `item.scores`, `write.result`

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Continuous Data

# Example 1a: Coefficient omega
item.omega(HolzingerSwineford1939, x1::x9)

# Example 1b: Coefficient omega and item statistics after excluding 'x3'
item.omega(HolzingerSwineford1939, x1::x9, exclude = "x3", print = "all")
```

```
# Example 2: Hierarchical Omega
item.omega(HolzingerSwineford1939, x1::x9, type = "hierarch")

# Example 3a: Residual covariance between 'x1' and 'x2'
item.omega(HolzingerSwineford1939, x1::x9, rescov = c("x1", "x2"))

# Example 3b: Residual covariances between 'x1' and 'x2', and 'x2' and 'x3'
item.omega(HolzingerSwineford1939, x1::x9, rescov = list(c("x1", "x2"), c("x2", "x3")))

# Example 4: Summary of the CFA model used to compute coefficient omega
lavaan::summary(item.omega(HolzingerSwineford1939, x1::x9, output = FALSE)$model.fit,
                standardized = TRUE)

#————————————————————————————————————————————————————————————————————
# Polytomous Data

# Example 5: Ordinal coefficient omega and item statistics
item.omega(data.items, pitem1, pitem2r, pitem3r, pitem4::pitem6, type = "categ",
           print = "all")

#————————————————————————————————————————————————————————————————————
# Write Results

# Example 6a: Write Results into a text file
item.omega(HolzingerSwineford1939, x1::x9, print = "all", write = "Omega.txt")

# Example 6b: Write Results into an Excel file
item.omega(HolzingerSwineford1939, x1::x9, print = "all", write = "Omega.xlsx")

## End(Not run)
```

`item.reverse` *Reverse Code Scale Item*

Description

This function reverse codes inverted items, i.e., items that are negatively worded.

Usage

```
item.reverse(data, ..., min = NULL, max = NULL, keep = NULL, append = TRUE,
             name = ".r", as.na = NULL, table = FALSE, check = TRUE)
```

Arguments

`data`	a numeric vector for reverse coding an item or data frame for reverse coding more than one item.
`...`	an expression indicating the variable names in `data` e.g., `item.reverse(x1, x2, x3, data = dat)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

min	an integer indicating the minimum of the item (i.e., lowest possible scale value).
max	an integer indicating the maximum of the item (i.e., highest possible scale value).
keep	a numeric vector indicating values not to be reverse coded.
append	logical: if TRUE (default), recoded variable(s) are appended to the data frame specified in the argument data.
name	a character vector indicating the names of the reverse coded item. By default, variables are named with the ending ".r" resulting in e.g. "x1.r" and "x2.r". Variable names can also be specified using a character vector matching the number of variables (e.g., name = c("reverse.x1", "reverse.x2")).
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
table	logical: if TRUE, a cross table item x reverse coded item is printed on the console if only one variable is specified.
check	logical: if TRUE (default), argument specification is checked.

Details

If the arguments min and/or max are not specified, empirical minimum and/or maximum is computed from the data. Note that reverse coding may fail if the lowest or highest possible scale value is not represented in the data. Therefore, it is always preferable to specify the arguments min and max.

Value

Returns a numeric vector or data frame with the same length or same number of rows as data containing the reverse coded scale item(s).

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

item.alpha, item.omega, rec, item.scores

Examples

```
# Example 1a: Reverse code 'pitem2' and append to 'data.items'
item.reverse(data.items, pitem2, min = 0, max = 3)

# Example 1b: Alternative specification without using the '...' argument
item.reverse(data.items$pitem2, min = 0, max = 3)

# Example 2a: Reverse code 'pitem2' and 'pitem3' and attach to 'data.items'
```

```
item.reverse(data.items, pitem2, pitem3, min = 0, max = 3)

# Example 2b: Alternative specification without using the '...' argument
dat <- cbind(data.items,
             item.reverse(data.items[, c("pitem2", "pitem3")], min = 0, max = 3))
```

---

item.scores *Scale Scores*

Description

This function computes (prorated) scale scores by averaging the (available) items that measure a single construct by default.

Usage

```
item.scores(data, ..., fun = c("mean", "sum", "median", "var", "sd", "min", "max"),
            prorated = TRUE, p.avail = NULL, n.avail = NULL, append = TRUE,
            name = "scores", as.na = NULL, check = TRUE)
```

Arguments

data	a data frame with numeric vectors.
...	an expression indicating the variable names in data, e.g., item.scores(dat, x1, x2, x3). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
fun	a character string indicating the function used to compute scale scores, default: "mean".
prorated	logical: if TRUE (default), prorated scale scores are computed (see 'Details'); if FALSE, scale scores of only complete cases are computed.
p.avail	a numeric value indicating the minimum proportion of available item responses needed for computing a prorated scale score for each case, e.g. p.avail = 0.8 indicates that scale scores are only computed for cases with at least 80% of item responses available. By default prorated scale scores are computed for all cases with at least one item response. Note that either argument p.avail or n.avail is used to specify the proration criterion.
n.avail	an integer indicating the minimum number of available item responses needed for computing a prorated scale score for each case, e.g. n.avail = 2 indicates that scale scores are only computed for cases with item responses on at least 2 items. By default prorated scale scores are computed for all cases with at least one item response. Note that either argument p.avail or n.avail is used to specify the proration criterion.
append	logical: if TRUE (default), a variable with scale scores is appended to the data frame specified in the argument data.

name	a character string indicating the names of the variable appended to the data frame specified in the argument data when append = TRUE. By default, the variable is named scores.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE (default), argument specification is checked.

Details

Prorated mean scale scores are computed by averaging the available items, e.g., if a participant answers 4 out of 8 items, the prorated scale score is the average of the 4 responses. Averaging the available items is equivalent to substituting the mean of a participant's own observed items for each of the participant's missing items, i.e., *person mean imputation* (Mazza, Enders & Ruehlman, 2015) or *ipsative mean imputation* (Schafer & Graham, 2002).

Proration may be reasonable when (1) a relatively high proportion of the items (e.g., 0.8) and never fewer than half are used to form the scale score, (2) means of the items comprising a scale are similar and (3) the item-total correlations are similar (Enders, 2010; Graham, 2009; Graham, 2012). Results of simulation studies indicate that proration is prone to substantial bias when either the item means or the inter-item correlation vary (Lee, Bartholow, McCarthy, Pederson & Sher, 2014; Mazza et al., 2015).

Value

Returns a numeric vector with the same length as nrow(x) containing (prorated) scale scores.

Author(s)

Takuya Yanagida

References

Enders, C. K. (2010). *Applied missing data analysis*. New York, NY: Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

Graham, J. W. (2012). Missing data: Analysis and design. New York, NY: Springer

Lee, M. R., Bartholow, B. D., McCarhy, D. M., Pederson, S. L., & Sher, K. J. (2014). Two alternative approaches to conventional person-mean imputation scoring of the self-rating of the effects of alcohol scale (SRE). *Psychology of Addictive Behaviors, 29*, 231-236. https://doi.org/10.1037/adb0000015

Mazza, G. L., Enders, C. G., & Ruehlman, L. S. (2015). Addressing item-level missing data: A comparison of proration and full information maximum likelihood estimation. *Multivariate Behavioral Research, 50*, 504-519. https://doi.org/10.1080/00273171.2015.1068157

Schafer, J. L., & Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods, 7*, 147-177.' https://doi.org/10.1037/1082-989X.7.2.147

See Also

cluster.scores, item.alpha, item.cfa, item.omega,

Examples

```
## Not run:

# Reverse code inverted item
data.items <- item.reverse(data.items, pitem2, pitem3, min = 0, max = 3)

# Example 1: Prorated mean scale scores
item.scores(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6)

# Example 2: Prorated standard deviation scale scores
item.scores(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6, fun = "sd")

# Example 3: Sum scale scores without proration
item.scores(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6, fun = "sum",
            prorated = FALSE)

# Example 4: Prorated mean scale scores,
# minimum proportion of available item responses = 0.8
item.scores(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6, p.avail = 0.8)

# Example 5: Prorated mean scale scores,
# minimum number of available item responses = 3
item.scores(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6, n.avail = 3)

## End(Not run)
```

`item.stats` *Item Statistics for Dichotomous, Polytomous, and Continuous Items*

Description

This function provides item statistics for dichotomous, polytomous, and continuous items by computing corrected item-total correlations with confidence intervals and coefficient alphas if the item is deleted. The default setting for computing the item-total correlations is depending on the type of the data, i.e., point-biserial correlation for dichotomous data, polyserial correlation for polytomous data, and product-moment correlation for continuous data.

Usage

```
item.stats(data, ..., exclude = NULL, correct = TRUE,
           method = c("pearson", "pbiser", "biser", "polyser"),
           adjust = c("none", "joint", "approx"),
           missing = c("listwise", "pairwise", "fiml"),
           alternative = c("two.sided", "less", "greater"),
           conf.level = 0.95, na.omit = FALSE, digits = 2, r.digits = 3,
           as.na = NULL, write = NULL, append = TRUE,
           check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. Note that dichotomous items must be coded with `0` and `1`, while polytomous should be coded either as ordered factors or as integer values. However, the function assumes continuous data whenever items have more than two distinct values. Polytomous items can be specified either by providing a data frame with ordered factors or by specifying a data frame with numeric vectors with integer values, along with setting the argument `method` to `polyser`.

`...` an expression indicating the variable names in `data` e.g., `item.stat(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`exclude` a character vector indicating items to be excluded from the analysis.

`correct` logical: if `TRUE` (default), the corrected item-total correlation is computed.

`method` a character string indicating which correlation coefficient is used for the item-total correlation, i.e., `"pearson"` for the Pearson product-moment correlation coefficient when data are continuous, `"pbiser"` for the point-biserial correlation coefficient when data are dichotomous, `"biser"` for the biserial correlation coefficient when data are dichotomous, and `"polyser"` for the polyserial correlation coefficient when data are polytomous. By default, the corrected item-total point-biserial correlation (`"pbiser"`) is computed for dichotomous items, corrected item-total polyserial correlation (`"polyser"`) is computed for polytomous items, and the corrected item-total product-moment correlation (`"pearson"`) is computed for continuous items.

`adjust` a character string specifying the non-normality adjustment method (see 'Details' in the `ci.cor` function), i.e., `"none"` for the Fisher z' confidence interval for the Pearson product-moment correlation coefficient without non-normality adjustment, `"joint"` for the confidence interval with non-normality adjustment via sample joint moments, and `"approx"` (default) for the confidence interval with non-normality adjustment via approximate distribution by skewness and kurtosis. Note that this argument only applies to the Pearson product-moment correlation coefficient, i.e., `method = "pearson"`.

`missing` a character string indicating how to deal with missing data when computing coefficient alphas if the item is deleted, i.e., `"listwise"` for listwise deletion, `"pairwise"` (default) for pairwise deletion, `"fiml"` for full information maximum likelihood method. Note that the argument `na.omit` switches to `TRUE` when specifying `missing = "listwise"`.

`alternative` a character string specifying the alternative hypothesis for the confidence intervals for the item-total correlation, i.e., `"two.sided"` (default), `"greater"` or `"less"`.

`conf.level` a numeric value between 0 and 1 indicating the confidence level of the interval for the item-total correlation.

`na.omit` logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion). By default pairwise deletion is used for computing point biserial and product-moment correlation, while listwise deletion is used for computing biserial and polyserial correlation. Note that the argument `missing` switches to `"listwise"` when specifying `na.omit = TRUE`.

digits	an integer value indicating the number of decimal places to be used to display display item difficulty, mean, median, and standard deviation.
r.digits	an integer value indicating the number of decimal places to be used to display item-total correlation, and coefficient alpha if the item is deleted.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Details

Confidence Interval for the Item-Total Correlation The confidence interval for the point-biserial correlation is based on Bonett (2020), while the confidence interval for the biserial and polyserial correlation is based on the maximum likelihood standard error provided by the function polyserial from the **polycor** package (Fox, 2025). The confidence interval for the Pearson product-moment correlation uses the ci.cor function from the **misty** package that computes a confidence interval with non-normality adjustment via approximate distribution by skewness and kurtosis by default.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame including all variables used in the analysis
dtype	type of the data, i.e., "dich" for dichotomous, "poly" for polytomous, and "cont" for continuous data
args	specification of function arguments
result	result table

Note

This function uses a modified copy of the polyserial() function in the **polycor** package by John Fox (2025) when requesting item-total biserial correlation.

Author(s)

Takuya Yanagida

References

Bonett D. G. (2020). Point-biserial correlation: Interval estimation, hypothesis testing, meta-analysis, and sample size determination. *The British Journal of Mathematical and Statistical Psychology, 73* Suppl 1, 113–144. https://doi.org/10.1111/bmsp.12189

Fox, J. (2025). *polycor: Polychoric and polyserial correlations*. R package version 0.8-2. https://doi.org/10.32614/CRAN.package.polycor

See Also

item.distract, item.alpha, item.omega, item.cfa, item.reverse, item.scores

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Dichotomous Data

# Example 1a: Corrected item-total point-biserial correlation
item.stats(data.items, +ditem)

# Example 1b: Corrected item-total biserial correlation
item.stats(data.items, +ditem, method = "biser")

# Example 1c: Uncorrected item-total point-biserial correlation
item.stats(data.items, +ditem, correct = FALSE)

# Example 1d: Display item-total correlation and coefficient alpha with 2 digits
item.stats(data.items, +ditem, r.digits = 2)

#————————————————————————————————————————————————————————————————————————
# Polytomous Data

# Reverse code inverted item
data.items <- item.reverse(data.items, pitem2, pitem3, min = 0, max = 3)

# Example 2a: Corrected item-total polyserial correlation
item.stats(data.items, pitem1, pitem2.r, pitem3.r, pitem4::pitem6, method = "polyser")

# Example 2b: One-sided confidence intervals
item.stats(data.items, +pitem, method = "polyser", alternative = "greater")

#————————————————————————————————————————————————————————————————————————
# Continuous Data

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

# Example 3a: Corrected item-total product-moment correlation
item.stats(HolzingerSwineford1939, x1::x9)
```

```
# Example 3a: Exclude items 'x2' and 'x7'
item.stats(HolzingerSwineford1939, x1::x9, exclude = c("x2", "x7"))

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 4a: Write Results into a text file
item.stats(data.items, +ditem, write = "Item-Stats.txt", output = FALSE)

# Example 4b: Write Results into an Excel file
item.stats(data.items, +ditem, write = "Item-Stats.xlsx", output = FALSE)

## End(Not run)
```

`lagged` *Create Lagged Variables*

Description

This function computes lagged values of variables by a specified number of observations. By default, the function returns lag-1 values of the vector or data frame specified in the first argument.

Usage

```
lagged(data, ..., id = NULL, obs = NULL, day = NULL, lag = 1, time = NULL,
       units = c("secs", "mins", "hours", "days", "weeks"), append = TRUE,
       name = ".lag", name.td = ".td", as.na = NULL, check = TRUE)
```

Arguments

`data`	a numeric vector for computing a lagged values for a variable or data frame for computing lagged values for more than one variable. Note that the subject ID variable (`id`), observation number variable (`obs`), day number variable (`day`), and the date and time variable (`time`) are excluded from `data` when specifying theses arguments.
`...`	an expression indicating the variable names in `data`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`id`	either a character string indicating the variable name of the subject ID variable or a vector representing the subject IDs, see 'Details'.
`obs`	either a character string indicating the variable name of the observation number variable or a vector representing the observations. Note that duplicated values within the same subject ID are not allowed, see 'Details'.
`day`	either a character string indicating the variable name of the day number variable in or a vector representing the days, see 'Details'.
`lag`	a numeric value specifying the lag, e.g. `lag = 1` (default) returns lag-1 values.

time	a variable of class POSIXct or POSIXlt representing the date and time of the observation used to compute time differences between observations.
units	a character string indicating the units in which the time difference is represented, i.e., "secs" for seconds, "mins" (default) for minutes, "hours" for hours, "days" for days, and "weeks" for weeks.
append	logical: if TRUE (default), lagged variable(s) are appended to the data frame specified in the argument data.
name	a character vector indicating the names of the lagged variables. By default, lagged variables are named with the ending ".lag" resulting in e.g. "x1.lag" and "x2.lag" when specifying two variables. Variable names can also be specified using a character vector matching the number of variables, e.g., name = c("lag.x1", "lag.x2")).
name.td	a character vector indicating the names of the time difference variables when specifying a date and time variables for the argument time. By default, time difference variables are named with the ending ".td" resulting in e.g. "x1.td" and "x2.td" when specifying two variables. Variable names can also be specified using a character vector matching the number of variables specified, e.g., name = c("td.x1", "td.x2")).
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to the argument data, but not to cluster.
check	logical: if TRUE (default), argument specification is checked.

Details

The function is used to create lagged version of the variable(s) specified via the data argument:

If the id argument is not specified i.e., id = NULL, all observations are assumed to come from the same subject. If the dataset includes multiple subjects, then this variable needs to be specified so that observations are not lagged across subjects

Optional argument day If the day argument is not specified i.e., day = NULL, values of the variable to be lagged are allowed to be lagged across days in case there are multiple observation days.

Optional argument obs If the obs argument is not specified i.e., obs = NULL, consecutive observations from the same subjects are assumed to be one lag apart.

Value

Returns a numeric vector or data frame with the same length or same number of rows as data containing the lagged variable(s).

Note

This function is a based on the lagvar() function in the **esmpack** package by Wolfgang Viechtbauer and Mihail Constantin (2023).

Author(s)

Takuya Yanagida

References

Viechtbauer W, Constantin M (2023). *esmpack: Functions that facilitate preparation and management of ESM/EMA data*. R package version 0.1-20.

See Also

center, rec, coding, item.reverse.

Examples

```
dat <- data.frame(subject = rep(1:2, each = 6),
                  day = rep(1:2, each = 3),
                  obs = rep(1:6, times = 2),
                  time = as.POSIXct(c("2024-01-01 09:01:00", "2024-01-01 12:05:00",
                                      "2024-01-01 15:14:00", "2024-01-02 09:03:00",
                                      "2024-01-02 12:21:00", "2024-01-02 15:03:00",
                                      "2024-01-01 09:02:00", "2024-01-01 12:09:00",
                                      "2024-01-01 15:06:00", "2024-01-02 09:02:00",
                                      "2024-01-02 12:15:00", "2024-01-02 15:06:00")),
                   pos = c(6, 7, 5, 8, NA, 7, 4, NA, 5, 4, 5, 3),
                   neg = c(2, 3, 2, 5, 3, 4, 6, 4, 6, 4, NA, 8))

# Example 1: Lagged variable for 'pos'
lagged(dat$pos, id = dat$subject, day = dat$day)

# Example 1b: Alternative specification without using the '...' argument
lagged(dat[, c("pos", "subject", "day")], id = "subject", day = "day")

# Example 1c: Alternative specification using the 'data' argument
lagged(pos, data = dat, id = "subject", day = "day")

# Example 2a: Lagged variable for 'pos' and 'neg'
lagged(dat[, c("pos", "neg")], id = dat$subject, day = dat$day)

# Example 2b: Alternative specification using the 'data' argument
lagged(pos, neg, data = dat, id = "subject", day = "day")

# Example 3: Lag-2 variables for 'pos' and 'neg'
lagged(pos, neg, data = dat, id = "subject", day = "day", lag = 2)

# Example 4: Lagged variable and time difference variable
lagged(pos, neg, data = dat, id = "subject", day = "day", time = "time")

# Example 5: Lagged variables and time difference variables,
# name variables
lagged(pos, neg, data = dat, id = "subject", day = "day", time = "time",
       name = c("p.lag1", "n.lag1"), name.td = c("p.diff", "n.diff"))
```

```
# Example 6: NA observations excluded from the data frame
dat.excl <- dat[!is.na(dat$pos), ]

# Number of observation not taken into account, i.e.,
# - observation 4 used as lagged value for observation 6 for subject 1
# - observation 1 used as lagged value for observation 3 for subject 2
lagged(pos, data = dat.excl, id = "subject", day = "day")

# Number of observation taken into account by specifying the 'ob' argument
lagged(pos, data = dat.excl, id = "subject", day = "day", obs = "obs")
```

`libraries` *Load and Attach Multiple Packages*

Description

This function loads and attaches multiple add-on packages at once.

Usage

```
libraries(..., install = FALSE, quiet = TRUE, check = TRUE, output = TRUE)
```

Arguments

`...`	the names of the packages to be loaded, given as names (e.g., `misty`, `lavaan`, `lme4`), or literal character strings (e.g., `"misty"`, `"lavaan"`, `"lme4"`), or character vector (e.g., `c("misty", "lavaan", "lme4")`).
`install`	logical: if `TRUE`, missing packages and dependencies are installed.
`quiet`	logical: if `TRUE` (default), startup messages when loading package are disabled.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: logical: if `TRUE`, output is shown on the console.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`library`, `require`

Examples

```
## Not run:

# Example 1: Load packages using the names of the packages
misty::libraries(misty, lme4, lmerTest)

# Example 2: Load packages using literal character strings
misty::libraries("misty", "lme4", "lmerTest")

# Example 3: Load packages using a character vector
misty::libraries(c("misty", "lme4", "lmerTest"))

# Example 4: Check packages, i.e., TRUE = all depends/imports/suggests installed
misty::libraries(misty, lme4, lmerTest, output = FALSE)$result$restab

# Example 5: Depends, FALSE = not installed, TRUE = installed
misty::libraries(misty, lme4, lmerTest, output = FALSE)$result$depends

# Example 6: Imports, FALSE = not installed, TRUE = installed
misty::libraries(misty, lme4, lmerTest, output = FALSE)$result$imports

# Example 6: Suggests, FALSE = not installed, TRUE = installed
misty::libraries(misty, lme4, lmerTest, output = FALSE)$result$suggests

## End(Not run)
```

`modcomp` *Model Comparison*

Description

This function performs model comparison by providing a table with fit indices for lavaan model objects, information criteria, and F-tests or likelihood ratio tests for models estimated by the function `cfa()`, `sem()`, `growth()`, `lavaan()` from the **lavaan** package, `lm()`, `glm()`, `nls()` from the **stats** package, `lmer()`, `glmer()`, `glmer.nb()` from the **lme4** package, `lme()`, `nlme()` from the **nlme** package, `glmmTMB()` from the **glmmTMB** package, `betareg` from the **betareg** package, or `glm.nb()` and `polr()` from the **MASS** package. By default, the function provides the fit indices CFI, TLI, RMSEA, and SRMR for lavaan model objects and the information criteria AIC, CAIC, BIC, and SABIC.

Usage

```
modcomp(..., difftest = FALSE,
       print.fit = c("none", "deviance", "chisq", "cfi", "tli", "rmsea", "srmr"),
        fit.robust = c("standard", "scaled", "robust"),
        print.ic = c("all", "default", "none", "aic", "caic", "bic", "sabic",
                     "aicc", "hqc", "hbic", "spbic", "ibic", "sic", "icomp"),
       fit.digits = 3, ic.digits = 0, p.digits = 3, write = NULL, append = TRUE,
        check = TRUE, output = TRUE)
```

Arguments

`...` a fitted model object or sequence of fitted model objects of class `"lavaan"`, `"lm"`, `"glm"`, `"nls"`, `"lmerMod"`, `"lmerModLmerTest"`, `"glmerMod"`, `"lme"`, `"nlme"`, `"glmmTMB"`, `"betareg"`, `"negbin"`, or `"polr"`.

`difftest` logical: if `TRUE`, results of the F-test, chi-square difference test, or likelihood ratio test are printed on the console. Note that the function does not provide difference tests for models fitted by using the `betareg()` function from the **betareg** package.

`print.fit` a character vector indicating which fit indices to be printed on the console when specifying lavaan objects for the argument `...`, i.e., `"none"` for no fit indices, `"deviance"` for the deviance (i.e., log-likelihood multiplied by -2), `"chisq"` for the chi-square value, `"cfi"` for the comparative fit index, `"tli"` for the Tucker-Lewis-index, `"rmsea"` for the root mean square error of approximation, and `"srmr"` for the standardized root mean squared residual. By default, all fit indices are printed on the console.

`fit.robust` a character string indicating which version of the CFI, TLI, and RMSEA to show on the console when using a robust estimation method involving a scaling correction factor for model estimation in lavaan, i.e., `"standard"` (default when `estimator` is one of `"ML"`, `"MLF"`, `"GLS"`, `"WLS"`, `"DWLS"`, or `"ULS"`) for fit indices without any non-normality correction, `"scaled"` (default when `estimator` is one of `"MLMVS"`, `"ULSM"`, `"ULSMV"`, `"DLS"`, or `"PML"`) for population-corrected robust fit indices with ad hoc non-normality correction, and `robust` (default when `estimator` is one of `"MLM"`, `"MLMV"`, `"MLR"`, `"WLSM"`, or `"WLSMV"`) for sample-corrected robust fit indices based on formula provided by Li and Bentler (2006) and Brosseau-Liard and Savalei (2014).

`print.ic` a character vector indicating which information criteria to be printed on the console, i.e., `"all"` for printing all information criteria, `"default"` for printing the default set of information criteria (i.e., AIC, CAIC, BIC, and SABIC), `"none"` for no information criteria, `"aic"` for the Akaike information criterion (AIC), `"caic"` for the Consistent Akaike's information criterion (CAIC), `"bic"` for the Bayesian information criterion (BIC), `"sabic"` for the sample-size adjusted BIC (SABIC), `"aicc"` for the corrected Akaike information criterion (AICc), `"hqc"` for the Hannan–Quinn criterion (HQC), `"hbic"` for the Haughton's BIC (HBIC), `"spbic"` for the scaled unit-information prior BIC (SPBIC), `"ibic"` for the information-matrix-based BIC, `"sic"` for the stochastic information criterion (SIC), and `"icomp"` for the Bozdogan information complexity (ICOMP) criterion. The default setting is `print.ic = c("aic", "caic", "bic", "sabic)}` when `specifying more than one model object for the argument \code{...}, otherwise the default setting is \code{print.ic = "none`. Note that `"spbic"`, `"ibic"`, `"sic"`, and `"icomp"` are only available for lavaan model objects.

`fit.digits` an integer value indicating the number of decimal places to be used for displaying fit indices when comparing lavaan models.

`ic.digits` an integer value indicating the number of decimal places to be used for displaying information criteria.

`p.digits` an integer value indicating the number of decimal places to be used for displaying the *p*-value in the F-test or chi-square difference test.

`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Information Criteria Information criteria are statistical measures that attempt to balance model fit and model complexity to compare competing models for model selection. Most information criteria are based on the log-likelihood with a penalty for complexity, and typically have the following form (Preacher & Yaremych, 2023):

$$D + f(q, N)$$

where D is a function of the model's log-likelihood at convergence, whereas f is a function of the number estimated parameters (q) and the sample size (N).

- The **Akaike Information Criterion** (AIC; Akaike, 1973) is defined as

  $$\text{AIC} = -2LL + 2q$$

  The AIC is an efficient information criterion, i.e., it will asymptotically choose whichever model minimizes the mean square error of prediction (Vrieze, 2012). However, the AIC is not consistent, i.e., it is expected to pick different models at different N's (Kuha, 2004). Accordingly, AIC is expected to select more complex models as N increases, while in relatively small samples, the penalty for complexity has a greater influence and simpler models are selected (Preacher & Yaremych, 2023).
- The **Consistent Akaike Information Criterion** (CAIC; Bozdogan, 1987) is defined as

  $$\text{CAIC} = -2LL + q(ln(N) + 1)$$

  The CAIC modifies the standard AIC to be asymptotically consistent, i.e., it is expected to pick the true model as sample size increases. However, the CAIC is not considered an efficient information criterion. Compared to the BIC, the CAIC has a higher penalty for model complexity making it more consistent but also less efficient than the BIC.
- The **Bayesian Information Criterion** (BIC; Schwarz, 1978) is defined as

  $$\text{BIC} = -2LL + q \cdot ln(N)$$

  The BIC is a consistent information criterion, i.e., it will select the true model with probability approach 1 as N increases based on the assumption that (a) the true model is under consideration, (b) the true model's dimension remains fixed as N increases, and (c) the number of parameters in the true model is finite (Vrieze, 2012). Accordingly, BIC tends to select more parsimonious models than AIC and is less subject to choosing more complex models as N increase because the penalty term increases with N.

- The **Sample-Size Adjusted Bayesian Information Criterion** (SABIC; Sclove, 1987) is defined as

$$\text{SABIC} = -2LL + q \cdot ln(\frac{N+2}{24})$$

The SABIC is a variant of the BIC that reduces the penalty for complex models and seems to perfom bettern than BIC when the sample size is small to moderate (Chen et al., 2017).

- The **Corrected Akaike Information Criterion** (AICc; Burnham & Anderson, 2003) is defined as

$$\text{AICc} = AIC + \frac{2q(q+1)}{N-q-1}$$

The AICc is a corrected version of the AIC for small sample sizes or when the number of parameters is large relative to the sample size. Note that as the sample size increases the AICc converges to the standard AIC.

- The **Hannan–Quinn Criterion** (HQC; Hannan & Quinn, 1979) is defined as

$$\text{HQC} = -2LL + 2q \log(\log N)$$

The HQC imposes a penalty that is stronger than AIC but weaker than BIC in large sample as the penalty function decreases with increasing sample size and is often used to select the order of autoregressive processes.

- The **Haughton Bayesian Information Criterion** (HBIC; Haughton, 1988) is defined as

$$\text{HBIC} = -2LL + q \log \frac{N}{2\pi}$$

The HBIC performed well in model selection in simulation studies for structural equation models and had the best overall performance among the investigated information criteria along with the SPBIC (Haughton et al., 1997; Bollen et al., 2014).

- The **Scaled Unit-Information Prior Bayesian Information Criterion** (SPBIC; Bollen et al., 2012) is defined as

$$\text{SPBIC}_{\text{Case 1}} = -2LL + q(1 - \frac{q}{\hat{\theta}'\text{FIM}\hat{\theta}}), \text{or}$$

$$\text{SPBIC}_{\text{Case 2}} = -2LL + \hat{\theta}'\text{FIM}\hat{\theta},$$

depending on whether the product of the vector of estimated model parameters ($\hat{\theta}$) and the observed information matrix (FIM) exceeds the number of estimated parameters (Case 1) or not (Case 2). The SPBIC performed well in model selection in a simulation study for structural equation models, had the best overall performance among the investigated information criteria along with the HBIC (Bollen et al., 2014), and exhibited a better performance along with the IBIC than BIC and HBIC when the sample size was small (Bollen et al., 2012).

- The **Information Matrix-Based Bayesian Information Criterion** (IBIC; Bollen et al., 2014) is defined as

$$\text{IBIC} = -2LL + q \log \frac{N}{2\pi} + \log \det \text{FIM}$$

The IBIC performed well in model selection in a simulation study for structural equation models (Bollen et al., 2014) and exhibited a better performance along with the SPBIC than BIC and HBIC when the sample size was small (Bollen et al., 2012).

- The **Stochastic Information Criterion** (SIC; Rissanen, 1989) is defined as

$$\text{SIC} = -2LL + q \log N + \log \det \text{FIM} = -2LL - \log \det \text{ACOV}$$

The SIC performed well relative to other information criteria in two simulation studies of structural equation models applied to behavior genetic models (Markon & Krueger, 2004).

- The **Information Complexity Criterion** (ICOMP; Bozdogan & Haughton, 1988) is defined as

$$\text{ICOMP} = -2LL + 2C(\hat{\Sigma}_{Model})$$

where C represents a complexity measure and $\hat{\Sigma}_{Model}$ represents the estimated covariance matrix of the parameter vector estimated by the model, i.e., inverse Fisher information matrix (see Akman, 2010). The ICOMP penalizes the covariance complexity of the model instead of the number of estimated parameters.

In practice, it may be sensible to choose information criteria that emphasize *consistency* or *efficiency* as consistency and efficiency cannot be maximized simultaneously (Claeskens & Hjort, 2008). More specifically, a criterion that emphasizes *efficiency* such as AIC should be used when prediction or cross-validation is important, whereas a criterion that emphasizes *consistency* such as BIC should be used when we want to identify a model that best approximates the truth. Note that there is no such thing as a correct model, there are models that cross-validate better than others, and there are models that better reflect the true data-generating process (Preacher & Yaremych, 2023).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`class`	object class of the models specified in the argument ...
`model`	models specified in the argument ...
`args`	specification of function arguments
`result`	result table

Note

The computation of AICc, HQC, HBIC, SPBIC, IBIC, SIC, and ICOMP are based on the `moreFitIndices` function from the **semTools** package by Terrence D. Jorgensen, Sunthud Pornprasertmanit, Alexander M. Schoemann, and Yves Rosseel.

Author(s)

Takuya Yanagida

References

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. N. Petrov & B. F. Csaki (Eds.), *Second International Symposium on Information Theory*, (pp. 267-281). Academiai Kiado.

Akman, O. (2010). Information complexity based modeling in the presence of length-biased sampling. *Journal of Statistical Theory and Practice, 4*(1), 45-55. https://doi.org/10.1080/15598608.2010.10411972

Bollen, K. A., Harden, J. J., Ray, S., & Zavisca, J. (2014). BIC and alternative Bayesian information criteria in the selection of structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal, 21*(1), 1–19. https://doi.org/10.1080/10705511.2014.856691

Bollen, K. A., Ray, S., Zavisca, J., & Harden, J. J. (2012). A comparison of Bayes factor approximation methods including two new methods. *Sociological Methods & Research, 41*(2), 294-324. https://doi.org/10.1177/00491241124523

Burnham, K., & Anderson, D. (2003). *Model selection and multimodel inference: A practical–theoretic approach*. Springer.

Brosseau-Liard, P. E., & Savalei, V. (2014) Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research, 49*, 460-470. https://doi.org/10.1080/00273171.2014.933697

Chen, Q., Luo, W., Palardy, G. J., Glaman, R., & McEnturff, A. (2017). The efficacy of common fit indices for enumerating classes in growth mixture models when nested data structure is ignored: A Monte Carlo study. *SAGE Open, 7*(1). https://doi:10.1177/2158244017700459

Claeskins, G., & Hjort, N. L. (2008). *Model selection and model averaging*. Cambridge University Press.

Hannan, E.J. and Quinn, B.G. (1979) The determination of the order of an autoregression. *Journal of the Royal Statistical Society, 41*, 190-195.

Haughton, D. M. A. (1988). On the choice of a model to fit data from an exponential family. *The Annals of Statistics, 16*(1), 342-355.

Haughton, D., Oud, J., & Jansen, R. (1997). Information and other criteria in structural equation model selection. *Communications in Statistics, Part B - Simulation and Computation, 26*(4), 1477-1516.

Kuha, J. (2004). AIC and BIC: Comparisons of assumptions and performance. *Sociological Methods & Research, 33*, 188-229.

Li, L., & Bentler, P. M. (2006). Robust statistical tests for evaluating the hypothesis of close fit of misspecified mean and covariance structural models. *UCLA Statistics Preprint #506*. University of California.

Markon, K. E., & Krueger, R. F. (2004). An empirical comparison of information-theoretic selection criteria for multivariate behavior genetic models. *Behavior Genetics, 34*, 593-610.

Preacher, K. K., & Yaremych, H. E. (2023). Model selection in structural equation modeling. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (2nd ed., pp. 206-222). The Guilford Press.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. World Scientific.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics, 6*(2), 461-464.

Sclove, L. (1987). Application of model-selection criteria to some problems in multivariate analysis. *Psychometrika, 52*(3), 333-343.

Jorgensen, T. D., Pornprasertmanit, S., Schoemann, A. M., & Rosseel, Y. (2025). semTools: Useful tools for structural equation modeling. R package version 0.5-7. Retrieved from https://CRAN.R-project.org/package=semTools

Vrieze, S.I. (2012) Model selection and psychological theory: A discussion of the differences between the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). *Psychological Methods, 17*, 228-243. https://doi.org/10.1037/a0027127

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————————
# lavaan Model Objects

# Load lavaan package
library(lavaan)

# Model specification
HS.model <- 'visual  =~ x1 + x2 + x3
             textual =~ x4 + x5 + x6
             speed   =~ x7 + x8 + x9'

# Model estimation
fit1 <- cfa(HS.model, data = HolzingerSwineford1939)
fit2 <- cfa(HS.model, data = HolzingerSwineford1939, orthogonal = TRUE)

# Example 1a: Model fit for model 'fit1', default setting
modcomp(fit1)

# Example 1b: Model comparison, default setting
modcomp(fit1, fit2)

# Example 1c: Model comparison, request likelihood ratio test
modcomp(fit1, fit2, difftest = TRUE)

# Example 1d: Model comparison, request default information criteria and AICc
modcomp(fit1, fit2, print.ic = c("default", "aicc"))

# Example 1e: Model comparison, request all information criteria
modcomp(fit1, fit2, print.ic = "all")

# Example 1f: Model fit indices, request all information criteria
modcomp(fit1, print.ic = "all")

#————————————————————————————————————————————————————————————————————————
# lm Model Objects

# Model estimation
fit1 <- lm(mpg ~ cyl, data = mtcars)
fit2 <- lm(mpg ~ cyl + disp, data = mtcars)

# Example 2: Model comparison, requested F test
modcomp(fit1, fit2, difftest = TRUE)
```

```
#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 3a: Write Results into a text file
modcomp(fit1, fit2, difftest = TRUE, write = "Model_Comparison.txt")

# Example 3b: Write Results into a Excel file
modcomp(fit1, fit2, difftest = TRUE, write = "Model_Comparison.xlsx")

## End(Not run)
```

`mplus` *Create, Run, and Print Mplus Models*

Description

This wrapper function creates a Mplus input file, runs the input file by using the `mplus.run()` function, and prints the Mplus output file by using the `mplus.print()` function.

Usage

```
mplus(x, file = "Mplus_Input.inp", data = NULL, comment = FALSE,
      replace.inp = TRUE, mplus.run = TRUE, show.out = FALSE,
      replace.out = c("always", "never", "modified"), Mplus = .detect.mplus(),
      print = c("all", "input", "result"),
      input = c("all", "default", "data", "variable", "define", "analysis",
                "model", "montecarlo", "mod.pop", "mod.cov", "mod.miss",
                "message"),
      result = c("all", "default", "summary.analysis.short",
                 "summary.data.short", "random.starts", "summary.fit",
                 "mod.est", "fit", "class.count", "classif", "mod.result",
                 "total.indirect"),
      exclude = NULL, variable = FALSE, not.input = TRUE, not.result = TRUE,
      write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`x`	a character string containing the Mplus input text.
`file`	a character string indicating the name of the Mplus input file with or without the file extension `.inp`, e.g., `"Mplus_Input.inp"` or `"Mplus_Input"`.
`data`	a matrix or data frame from which the variables names for the subsection `NAMES` are extracted.
`comment`	logical: if `FALSE` (default), comments (i.e., text after the `!` symbol) are removed from the input text specified in the argument `x`.
`replace.inp`	logical: if `TRUE` (default), an existing input file will be replaced.

mplus.run	logical: if TRUE, the input file specified in the argument file containing the input text specified in the argument x is run using the mplus.run() function.
show.out	logical: if TRUE, estimation output (TECH8) is show on the R console. Note that if run within Rgui, output will display within R, but if run via Rterm, a separate window will appear during estimation.
replace.out	a character string for specifying three settings: "always" (default), which runs all models, regardless of whether an output file for the model exists, "never", which does not run any model that has an existing output file, and "modified", which only runs a model if the modified date for the input file is more recent than the output file modified date.
Mplus	a character string for specifying the name or path of the Mplus executable to be used for running models. This covers situations where Mplus is not in the system's path, or where one wants to test different versions of the Mplus program. Note that there is no need to specify this argument for most users since it has intelligent defaults.
print	a character vector indicating which results to show, i.e. "all" (default) for all results "input" for input command sections, and "result" for result sections.
input	a character vector specifying Mplus input command sections included in the output (see 'Details' in the mplus.print function).
result	a character vector specifying Mplus result sections included in the output (see 'Details' in the mplus.print function).
exclude	a character vector specifying Mplus input command or result sections excluded from the output (see 'Details' in the mplus.print function).
variable	logical: if TRUE, names of the variables in the data set (NAMES ARE) specified in the VARIABLE: command section are shown. By default, names of the variables in the data set are excluded from the output unless all variables are used in the analysis (i.e., no USEVARIABLES option specified in the Mplus input file).
not.input	logical: if TRUE (default), character vector indicating the input commands not requested are shown on the console.
not.result	logical: if TRUE (default), character vector indicating the result sections not requested are shown on the console.
write	a character string naming a file for writing the output into a text file with file extension ".txt" (e.g., "Output.txt").
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console by using the function mplus.print().

Details

The NAMES Option The NAMES option in the VARIABLE section used to assign names to the variables in the data set can be specified by using the data argument:

- Write Mplus Data File: In the first step, the Mplus data file is written by using the write.mplus() function, e.g. write.mplus(ex3_1, file = "ex3_1.dat").

- Specify Mplus Input: In the second step, the Mplus input is specified as a character string. The NAMES option is left out from the Mplus input text, e.g., input <- 'DATA: FILE IS ex3_1.dat;\nMODEL: y1 ON x1 x3;'.
- Run Mplus Input: In the third step, the Mplus input is run by using the mplus() function. The argument data needs to be specified given that the NAMES option was left out from the Mplus input text in the previous step, e.g., mplus(input, file = "ex3_1.inp", data = ex3_1).

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	a character vector containing the Mplus input text
args	specification of function arguments
input	list with input command sections
write	write command sections
result	list with input command sections (input) and result sections (result)

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.mplus, write.mplus, mplus.update, mplus.print, mplus.plot, mplus.bayes, mplus.run, mplus.lca

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Example 1: Write data, specify input, and run input

# Write Mplus Data File
write.mplus(ex3_1, file = "ex3_1.dat")

# Specify Mplus input, specify NAMES option
input1 <- '
DATA:     FILE IS ex3_1.dat;
VARIABLE: NAMES ARE y1 x1 x3;
MODEL:    y1 ON x1 x3;
OUTPUT:   SAMPSTAT;
```

```
'

# Run Mplus input
mplus(input1, file = "ex3_1.inp")

#————————————————————————————————————————————————————————————————————————
# Example 2: Alternative specification using the data argument

# Specify Mplus input, leave out the NAMES option
input2 <- '
DATA:     FILE IS ex3_1.dat;
MODEL:    y1 ON x1 x3;
OUTPUT:   SAMPSTAT;
'

# Run Mplus input, specify the data argument
mplus(input2, file = "ex3_1.inp", data = ex3_1)

## End(Not run)
```

`mplus.bayes` *Mplus Summary Measures, Convergence and Efficiency Diagnostics*

Description

This function uses the `h5file` function in the **hdf5r** package to read a Mplus GH5 file that is requested by the command `PLOT: TYPE IS PLOT2` in Mplus to compute point estimates (i.e., mean, median, and MAP), measures of dispersion (i.e., standard deviation and mean absolute deviation), measures of shape (i.e., skewness and kurtosis), credible intervals (i.e., equal-tailed intervals and highest density interval), convergence and efficiency diagnostics (i.e., potential scale reduction factor R-hat, effective sample size, and Monte Carlo standard error), probability of direction, and probability of being in the region of practical equivalence for the posterior distribution for each parameter. By default, the function computes the maximum of rank-normalized split-R-hat and rank normalized folded-split-R-hat, Bulk effective sample size (Bulk-ESS) for rank-normalized values using split chains, tail effective sample size (Tail-ESS) defined as the minimum of the effective sample size for 0.025 and 0.975 quantiles, the Bulk Monte Carlo standard error (Bulk-MCSE) for the median and Tail Monte Carlo standard error (Tail-MCSE) defined as the maximum of the MCSE for 0.025 and 0.975 quantiles.

Usage

```
mplus.bayes(x,
            print = c("all", "default", "m", "med", "map", "sd", "mad",
                      "skew", "kurt", "eti", "hdi",
                      "rhat", "b.ess", "t.ess", "b.mcse", "t.mcse"),
            param = c("all", "on", "by", "with", "inter", "var", "r2", "new"),
            std = c("all", "none", "stdyx", "stdy", "std"),
            m.bulk = FALSE, split = TRUE, rank = TRUE, fold = TRUE,
```

```
pd = FALSE, null = 0, rope = NULL,
ess.tail = c(0.025, 0.975), mcse.tail = c(0.025, 0.975),
alternative = c("two.sided", "less", "greater"),
conf.level = 0.95, digits = 2, r.digits = 3, ess.digits = 0,
mcse.digits = 3, p.digits = 3, write = NULL, append = TRUE,
check = TRUE, output = TRUE)
```

Arguments

`x` a character string indicating the name of the Mplus GH5 file (HDF5 format) with or without the file extension `.gh5`, e.g., `"Mplus_Plot.gh5"` or `"Mplus_Plot"`.

`print` a character vector indicating which summary measures, convergence, and efficiency diagnostics to be printed on the console, i.e. `"all"` for all summary measures, convergence, and efficiency diagnostics, `"m"` for the mean, `"med"` for the median, `"MAP"` for the maximum a posteriori probability estimate, `"med"` for the standard deviation, `"mad"` for the mean absolute deviation, `"skew"` for the skewness, `"kurt"` for the kurtosis, `"eti"` for the equal-tailed credible interval, `"hdi"` for the highest density credible interval, `"rhat"` for the potential scale reduction (PSR) factor R-hat convergence diagnostic, `"b.ess"` for the bulk effective sample size (ESS), `"t.ess"` for the tail ESS, `"b.mcse"` for the bulk Monte Carlo standard error (MCSE), and `"t.mcse"` for the tail MCSE. The default setting is `print = c("med", "sd", "skew", "kurt", "eti", "rhat", "b.ess", "t.ess", "b.mcse", "t.mcse")`.

`param` character vector indicating which parameters to print for the summary measures, convergence, and efficiency diagnostics, i.e., `"all"` for all parameters, `"on"` (default), for regression slopes, `"by"` for factor loadings, `"with"` for covariances, `"inter"` for intercepts and thresholds, `"var"` for (residual) variances, `"r2"` for r-square, and `"new"` for parameters not in the analysis model specified in the `NEW` option. The default setting is `"on"` if regression slopes are available. Otherwise, the default setting switches to `"by"` and to `"with"` if factor loadings are not available.

`std` a character vector indicating the standardized parameters to print for the summary measures, convergence, and efficiency diagnostics, i.e., `"all"` for all standardized parameters, `"none"` (default) for not printing any standardized parameters, `"stdyx"` for StdYX standardized parameters, `"stdy"` for StdY standardized parameters, and `"std"` for StdX standardized parameters.

`m.bulk` logical: if `TRUE` the Monte Carlo standard error for the mean is computed. The default setting is `m.bulk = FALSE`, i.e., the Monte Carlo standard error for the median is computed.

`split` logical: if `TRUE` (default), each MCMC chain is split in half before computing R-hat. Note that the argument `split` is always set to `FALSE` when computing ESS.

`rank` logical: if `TRUE` (default), rank-normalization is applied to the posterior draws before computing R-hat and ESS. Note that the argument `rank` is always set to `FALSE` when computing MCSE.

`fold` logical: if `TRUE` (default), the maximum of rank-normalized split-R-hat and rank normalized folded-split-R-hat is computed. Note that the arguments `split` and

rank are always set to TRUE when specifying fold = TRUE.

pd — logical: if TRUE, the probability of direction is printed on the console.

null — a numeric value considered as a null effect for the probability of direction (default is 0). Note that the value specified in the argument null applies to all parameters which might not be sensible for all parameters.

rope — a numeric vector with two elements indicating the ROPE's lower and upper bounds. ROPE is also depending on the argument alternative, e.g., if rope = c(-0.1, 0.1), then the actual ROPE is [-0.1, 0.1] given alternative = "two.sided} (default), \code{[-Inf, 0.1]} given \code{alternative = "greater, and [-0.1, Inf] given alternative = "less". Note that the interval specified in the argument rope applies to all parameters which might not be sensible for all parameters.

ess.tail — a numeric vector with two elements to specify the quantiles for computing the tail ESS. The default setting is tail = c(0.025, 0.975), i.e., tail ESS is the minimum of effective sample sizes for 5% and 95% quantiles.

mcse.tail — a numeric vector with two elements to specify the quantiles for computing the tail MCSE. The default setting is tail = c(0.025, 0.975), i.e., tail MCSE is the maximum of Monte Carlo standard error for 5% and 95% quantiles.

alternative — a character string specifying the alternative hypothesis for the credible intervals, must be one of "two.sided" (default), "greater" or "less".

conf.level — a numeric value between 0 and 1 indicating the confidence level of the credible interval. The default setting is conf.level = 0.95.

digits — an integer value indicating the number of decimal places to be used for displaying point estimates, measures of dispersion, and credible intervals.

r.digits — an integer value indicating the number of decimal places to be used for displaying R-hat values.

ess.digits — an integer value indicating the number of decimal places to be used for displaying effective sample sizes.

mcse.digits — an integer value indicating the number of decimal places to be used for displaying Monte Carlo standard errors.

p.digits — an integer value indicating the number of decimal places to be used for displaying the probability of direction and the probability of being in the region of practical equivalence (ROPE).

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification is checked.

output — logical: if TRUE (default), output is shown on the console by using the function mplus.print().

Details

Convergence and Efficiency Diagnostics for Markov Chains Convergence and efficiency diagnostics for Markov chains is based on following numeric measures:

- **Potential Scale Reduction (PSR) factor R-hat**: The PSR factor R-hat compares the between- and within-chain variance for a model parameter, i.e., R-hat larger than 1 indicates that the between-chain variance is greater than the within-chain variance and chains have not mixed well. According to the default setting, the function computes the improved R-hat as recommended by Vehtari et al. (2020) based on rank-normalizing (i.e., `rank = TRUE`) and folding (i.e., `fold = TRUE`) the posterior draws after splitting each MCMC chain in half (i.e., `split = TRUE`). The traditional R-hat used in Mplus can be requested by specifying `split = FALSE`, `rank = FALSE`, and `fold = FALSE`. Note that the traditional R-hat can catch many problems of poor convergence, but fails if the chains have different variances with the same mean parameter or if the chains have infinite variance with one of the chains having a different location parameter to the others (Vehtari et al., 2020). According to Gelman et al. (2014) a R-hat value of 1.1 or smaller for all parameters can be considered evidence for convergence. The Stan Development Team (2024) recommends running at least four chains and a convergence criterion of less than 1.05 for the maximum of rank normalized split-R-hat and rank normalized folded-split-R-hat. Vehtari et al. (2020), however, recommended to only use the posterior samples if R-hat is less than 1.01 because the R-hat can fall below 1.1 well before convergence in some scenarios (Brooks & Gelman, 1998; Vats & Knudon, 2018).
- **Effective Sample Size (ESS)**: The ESS is the estimated number of independent samples from the posterior distribution that would lead to the same precision as the autocorrelated samples at hand. According to the default setting, the function computes the ESS based on rank-normalized split-R-hat and within-chain autocorrelation. The function provides the estimated Bulk-ESS (`B.ESS`) and the Tail-ESS (`T.ESS`). The Bulk-ESS is a useful measure for sampling efficiency in the bulk of the distribution (i.e, efficiency of the posterior mean), and the Tail-ESS is useful measure for sampling efficiency in the tails of the distribution (e.g., efficiency of tail quantile estimates). Note that by default, the Tail-ESS is the minimum of the effective sample sizes for 5% and 95% quantiles (`tail = c(0.025, 0.975)`). According to Kruschke (2015), a rank-normalized ESS greater than 400 is usually sufficient to get a stable estimate of the Monte Carlo standard error. However, a ESS of at least 1000 is considered optimal (Zitzmann & Hecht, 2019).
- **Monte Carlo Standard Error (MCSE)**: The MCSE is defined as the standard deviation of the chains divided by their effective sample size and reflects uncertainty due to the stochastic algorithm of the Markov Chain Monte Carlo method. The function provides the estimated Bulk-MCSE (`B.MCSE`) for the margin of error when using the MCMC samples to estimate the posterior mean and the Tail-ESS (`T.MCSE`) for the margin of error when using the MCMC samples for interval estimation.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	Mplus GH5 file

`args`	specification of function arguments
`data`	three-dimensional array parameter x iteration x chain of the posterior
`result`	result table with summary measures, convergence, and efficiency diagnostics

Note

This function is a modified copy of functions provided in the **rstan** package by Stan Development Team (2024) and **bayestestR** package by Makowski et al. (2019).

Author(s)

Takuya Yanagida

References

Brooks, S. P. and Gelman, A. (1998). General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics, 7*(4): 434–455. MR1665662.

Gelman, A., & Rubin, D.B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science, 7*, 457-472. https://doi.org/10.1214/ss/1177011136

Kruschke, J. (2015). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

Makowski, D., Ben-Shachar, M., & Lüdecke, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the Bayesian framework. *Journal of Open Source Software, 4*(40), 1541. https://doi.org/10.21105/joss.01541

Stan Development Team (2024). *RStan: the R interface to Stan*. R package version 2.32.6. https://mc-stan.org/.

Vats, D. and Knudson, C. (2018). Revisiting the Gelman-Rubin Diagnostic. arXiv:1812.09384.

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2020). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian analysis, 16*(2), 667-718. https://doi.org/110.1214/20-BA1221

Zitzmann, S., & Hecht, M. (2019). Going beyond convergence in Bayesian estimation: Why precision matters too and how to assess it. *Structural Equation Modeling: A Multidisciplinary Journal, 26*(4), 646–661. https://doi.org/10.1080/10705511.2018.1545232

See Also

`read.mplus`, `write.mplus`, `mplus`, `mplus.update`, `mplus.print`, `mplus.plot`, `mplus.run`, `mplus.lca`

Examples

```
## Not run:

#-----------------------------------------------------------------------------
# Mplus Example 3.18: Moderated Mediation with a Plot of the Indirect Effect

# Example 1: Default setting
mplus.bayes("ex3.18.gh5")
```

```
# Example 2: Print all parameters
mplus.bayes("ex3.18.gh5", param = "all")

# Example 3: Print parameters not in the analysis model
mplus.bayes("ex3.18.gh5", param = "new")

# Example 4a: Print all summary measures, convergence, and efficiency diagnostics
mplus.bayes("ex3.18.gh5", print = "all")

# Example 4a: Print default measures plus MAP
mplus.bayes("ex3.18.gh5", print = c("default", "map"))

# Example 5: Print traditional R-hat in line with Mplus
mplus.bayes("ex3.18.gh5", split = FALSE, rank = FALSE, fold = FALSE)

# Example 6: Print probability of direction and the probability of
# being ROPE [-0.1, 0.1]
mplus.bayes("ex3.18.gh5", pd = TRUE, rope = c(-0.1, 0.1))

# Example 7: Write Results into a text file
mplus.bayes("ex3.18.gh5", write = "Bayes_Summary.txt")

# Example 8b: Write Results into a Excel file
mplus.bayes("ex3.18.gh5", write = "Bayes_Summary.xlsx")

## End(Not run)
```

`mplus.lca` *Mplus Model Specification and Estimation for Latent Class Analysis*

Description

This function writes Mplus input files for conducting latent class analysis (LCA) for continuous, count, ordered categorical, and unordered categorical variables. LCA with continuous indicator variables are based on six different variance-covariance structures, while LCA for all other variable types assume local independence. By default, the function conducts LCA with continuous variables and creates folders in the current working directory for each of the six sets of analysis, writes Mplus input files for conducting LCA with $k = 1$ to $k = 6$ classes into these folders, and writes the matrix or data frame specified in `x` into a Mplus data file in the current working directory. Optionally, all models can be estimated by setting the argument `mplus.run` to `TRUE`.

Usage

```
mplus.lca(x, ind = NULL,
          type = c("continuous", "count", "categorical", "nominal"),
          classes = 6, cluster = NULL,
          folder = c("A_Invariant-Theta_Diagonal-Sigma",
                     "B_Varying-Theta_Diagonal-Sigma",
```

```
                    "C_Invariant-Theta_Invariant-Unrestrictred-Sigma",
                    "D_Invariant-Theta_Varying-Unrestricted-Sigma",
                    "E_Varying-Theta_Invariant-Unrestricted-Sigma",
                    "F_Varying-Theta_Varying-Unrestricted-Sigma"),
        file = "Data_LCA.dat", missing = -99,
        write = c("all", "folder", "data", "input"),
        useobservations = NULL, estimator = "MLR",
        starts = c(100, 50), stiterations = 10, processors = c(8, 8),
        boot = c("none", "perc", "bc"), nrep = 1000,
        lrtbootstrap = 1000, lrtstarts = c(0, 0, 100, 50),
    output = c("all", "SVALUES", "CINTERVAL", "TECH7", "TECH8", "TECH11", "TECH14"),
        replace.inp = FALSE, mplus.run = FALSE, Mplus = "Mplus",
        replace.out = c("always", "never", "modified"), check = TRUE)
```

Arguments

`x` a matrix or data frame. Note that all variable names must be no longer than 8 character.

`ind` a character vector indicating the variables names of the latent class indicators in `x`.

`type` a character string indicating the variable type of the latent class indicators, i.e., `"continuous"` (default) for continuous variables, `"count"` for count variables, `"categorical"` for binary or ordered categorical variables, and `"nominal"` for unordered categorical variables. Note that it is not possible to mix different variable types in the analysis.

`classes` an integer value specifying the maximum number of classes for the latent class analysis. By default, LCA with a maximum of 6 classes is specified (i.e., $k = 1$ to $k = 6$).

`cluster` a character string indicating the cluster variable in the matrix or data frame specified in `x` representing the nested grouping structure for computing cluster-robust standard errors. Note that specifying a cluster variables does not have any effect on the information criteria, but on the Vuong-Lo-Mendell-Rubin likelihood ratio test of model fit.

`folder` a character vector with six character strings for specifying the names of the six folder representing different variance-covariance structures for conducting LCA with continuous indicator variables. There is only one folder for LCA with all other variable types which is called `"LCA_1-x_Classes"` with `x` being the maximum number of classes specified in the argument `classes`.

`file` a character string naming the Mplus data file with or without the file extension '.dat', e.g., `"Data_LCA.dat"` (default) or `"Data_LCA"`.

`missing` a numeric value or character string representing missing values (`NA`) in the Mplus data set. This values or character string will be specified in the Mplus input file as `MISSING IS ALL(missing)`. By default, `-99` is used to represent missing values.

`write` a character vector indicating whether to create the six folders specified in the argument `folder` (`"folder"`), to write the matrix or data frame specified in `x`

into a Mplus data file ("data"), and write the Mplus input files into the six folders specified in the argument folder ("input"). By default, the function creates the folders, writes the Mplus data file, and writes the Mplus input files into the folders.

useobservations
: a character string indicating the conditional statement to select observations.

estimator
: a character string for specifying the ESTIMATOR option in Mplus. By default, the estimator "MLR" is used.

starts
: a vector with two integer values for specifying the STARTS option in Mplus. The first number represents the number of random sets of starting values to generate in the initial stage and the second number represents the optimizations to use in the final stage. By default, 500 random sets of starting values are generated and 100 optimizations are carried out in the final stage.

stiterations
: an integer value specifying the STITERATIONS option in Mplus. The numeric value represents the maximum number of iterations allowed in the initial stage. By default, 50 iterations are requested.

processors
: a vector of one or two integer values for specifying the PROCESSORS option in Mplus. The values specifies the number of processors and threads to be used for parallel computing to increase computational speed. By default, 8 processors and threads are used for parallel computing.

boot
: a character string specifying the type of bootstrap confidence intervals (CI), i.e., "none" (default) for not conducting bootstrapping, "perc", for the percentile bootstrap CI (i.e., Mplus command OUTPUT: CINTERVAL (BOOTSTRAP)), and "bc" for the bias-corrected (BC) percentile bootstrap CI (i.e., Mplus command OUTPUT: CINTERVAL (BCBOOTSTRAP)).

nrep
: a numeric value indicating the number of bootstrap replicates (default is 1000).

lrtbootstrap
: an integer value for specifying the LRTBOOTSTRAP option in Mplus when requesting a parametric bootstrapped likelihood ratio test (i.e., output = "TECH14"). The value represents the number of bootstrap draws to be used in estimating the *p*-value of the parametric bootstrapped likelihood ratio test. By default, 1000 bootstrap draws are requested.

lrtstarts
: a vector with four integer values for specifying the LRTSTARTS option in Mplus when requesting a parametric bootstrapped likelihood ratio test (i.e., output = "TECH14"). The values specify the number of starting values to use in the initial stage and the number of optimizations to use in the final stage for the k - 1 and k classes model when the data generated by bootstrap draws are analyzed. By default, 0 random sets of starting values in the initial stage and 0 optimizations in the final stage are used for the k - 1 classes model and 100 random sets of starting values in the initial stage and 50 optimizations in the final stage are used for the k class model.

output
: a character vector specifying the TECH options in the OUTPUT section in Mplus, i.e., SVALUES to request input statements that contain parameter estimates from the analysis, CINTERVAL to request confidence intervals, TECH7 to request sample statistics for each class using raw data weighted by the estimated posterior probabilities for each class, TECH8 to request the optimization history in estimating the model, TECH11 to request the Lo-Mendell-Rubin likelihood ratio test of

model fit, and TECH14 to request a parametric bootstrapped likelihood ratio test. By default, SVALUES, CINTERVAL, and TECH11 are requested. Note that TECH11 is only available for the MLR estimator.

`replace.inp` logical: if TRUE, all existing input files in the folder specified in the argument `folder` are replaced.

`mplus.run` logical: if TRUE, all models in the folders specified in the argument `folder` are estimated by using the `mplus.run` function in the R package `misty`.

`Mplus` a character string for specifying the name or path of the Mplus executable to be used for running models. This covers situations where Mplus is not in the system's path, or where one wants to test different versions of the Mplus program. Note that there is no need to specify this argument for most users since it has intelligent defaults.

`replace.out` a character string for specifying three settings, i.e., `"always"` to run all models regardless of whether an output file for the model exists, `"never"` to not run any model that has an existing output file, and `"modified"` (default) to only runs a model if the modified date for the input file is more recent than the output file modified date.

`check` logical: if TRUE (default), argument specification is checked.

Details

Latent class analysis (LCA) is a model-based clustering and classification method used to identify qualitatively different classes of observations which are unknown and must be inferred from the data. LCA can accommodate continuous, count, binary, ordered categorical, and unordered categorical indicators. LCA with continuous indicator variables are also known as latent profile analysis (LPA). In LPA, the within-profile variance-covariance structures represent different assumptions regarding the variance and covariance of the indicator variables both within and between latent profiles. As the best within-profile variance-covariance structure is not known a priori, all of the different structures must be investigated to identify the best model (Masyn, 2013). This function specifies six different variance-covariance structures labeled A to F (see Table 1 in Patterer et al, 2023):

Model A The within-profile variance is constrained to be profile-invariant and covariances are constrained to be 0 in all profiles (i.e., equal variances across profiles and no covariances among indicator variables). This is the default setting in Mplus.

Model B The within-profile variance is profile-varying and covariances are constrained to be 0 in all profiles (i.e., unequal variances across profiles and no covariances among indicator variables).

Model C The within-profile variance is constrained to be profile-invariant and covariances are constrained to be equal in all profiles (i.e., equal variances and covariances across profiles).

Model D The within-profile variance is constrained to be profile-invariant and covariances are profile-varying (i.e., equal variances across profiles and unequal covariances across profiles).

Model E The within-profile variances are profile-varying and covariances are constrained to be equal in all profiles (i.e., unequal variances across profiles and equal covariances across profiles).

Model F The within-class variance and covariances are both profile-varying (i.e., unequal variances and covariances across profiles).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`x`	matrix or data frame specified in the argument x
`args`	specification of function arguments
`result`	list with six entries for each of the variance-covariance structures and Mplus inputs based on different number of profiles in case of continuous indicators or list of Mplus inputs based on different number of classes in case of count, ordered or unordered categorical indicators.

Author(s)

Takuya Yanagida

References

Masyn, K. E. (2013). Latent class analysis and finite mixture modeling. In T. D. Little (Ed.), *The Oxford handbook of quantitative methods: Statistical analysis* (pp. 551–611). Oxford University Press.

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

Patterer, A. S., Yanagida, T., Kühnel, J., & Korunka, C. (2023). Daily receiving and providing of social support at work: Identifying support exchange patterns in hierarchical data. *Journal of Work and Organizational Psychology, 32*(4), 489-505. https://doi.org/10.1080/1359432X.2023.2177537

See Also

`mplus.lca.summa`, `read.mplus`, `write.mplus`, `mplus`, `mplus.update`, `mplus.print`, `mplus.plot`, `mplus.bayes`, `mplus.run`

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Example 1: LCA with k = 1 to k = 8 profiles, continuous indicators
# Input statements that contain parameter estimates
# Vuong-Lo-Mendell-Rubin LRT and bootstrapped LRT
mplus.lca(HolzingerSwineford1939, ind = c("x1", "x2", "x3", "x4"),
          classes = 8, output = c("SVALUES", "TECH11", "TECH14"))

#————————————————————————————————————————————————————————————————————————
# Example 22: LCA with k = 1 to k = 6 profiles, ordered categorical indicators
# Select observations with ageyr <= 13
```

```
# Estimate all models in Mplus
mplus.lca(round(HolzingerSwineford1939[, -5]), ind = c("x1", "x2", "x3", "x4"),
          type = "categorical", useobservations = "ageyr <= 13",
          mplus.run = TRUE)

## End(Not run)
```

`mplus.lca.summa` *Summary Result Tables and Grouped Bar Charts for Latent Class Analysis in Mplus*

Description

This function reads all Mplus output files from latent class analysis in subfolders to create result tables with model summaries (e.g., AIC, CAIC, BIC, SABIC, AWE and cmP), approximate Bayes factors, classification diagnostics (e.g., relative Entropy, AvePP, and OCC), class-specific means and variances or class-specific item response probabilities of the indicator variables, and Cohen's *d*s to quantify class separation between latent class *j* and latent class *k*. By default, the function reads output files in all subfolders of the current working directory or output files in the current working directory and prints a table with model summaries on the console. Bar charts including confidence intervals for each latent class solution can be requested by setting the argument `plot` to `TRUE`. Note that result tables with Bayes factors, classification diagnostics, class-specific means and variances, class-specific item response probabilities, and Cohen's *d*s will not be printed on the console, but are only available in the exported Excel file when specifying the `write` argument (e.g., `write = "Results_LCA.xlsx"`).

Usage

```
mplus.lca.summa(folder = getwd(), exclude = NULL, sort.n = TRUE, sort.p = FALSE,
                digits = 0, p.digits = 3, bf.trunc = TRUE, conf.level = 0.95,
                plot = FALSE, group.ind = TRUE, ci = TRUE,
                axis.title = 9, axis.text = 9, levels = NULL, labels = NULL,
               ylim = NULL, ylab = c("Mean Value", "Item Response Probability"),
                breaks = ggplot2::waiver(), errorbar.width = 0.1,
                legend.title = 9, legend.text = 9, legend.key.size = 0.5,
                gray = FALSE, start = 0.15, end = 0.85, dpi = 600,
                width.ind = NULL, width.nclass = NULL, height.categ = NULL,
                height = NA, write = NULL, append = TRUE, check = TRUE,
                output = TRUE)
```

Arguments

`folder`	a character string indicating the path of the folder containing subfolders with the Mplus output files. By default Mplus outputs in the subfolders of the current working directory are read. Note that if there are no subfolders available, Mplus outputs from the folder specified in the argument `folder` are extracted.
`exclude`	a character vector indicating the name of the subfolders excluded from the result tables.

`sort.n`	logical: if `TRUE` (default), result table is sorted according to the number of classes within each folder in increasing order.
`sort.p`	logical: if `TRUE`, model-estimated class counts, proportions, average posterior class probabilities, and odds of correct classification ratio in the classification diagnostics are sorted in increasing order.
`digits`	an integer value indicating the number of decimal places to be used for displaying LL, AIC, CAIC, BIC, SABIC, AWE, OCC, and approximate Bayes factors (aBF).
`p.digits`	an integer value indicating the number of decimal places to be used for displaying cmP, *p*-values, relative entropy values, class proportions, and confidence intervals. Note that the scaling correction factor is displayed with `p.digits` minus 1 decimal places.
`bf.trunc`	logical: if `TRUE` (default), approximate Bayes factors (aBF) greater than 1000 are truncated.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the intervals in the result table with means and variances for each latent class separately. Note that only `0.9`, `0.95`, or `0.95` are allowed when extracting confidence intervals from Mplus outputs.
`plot`	logical: if `TRUE`, bar charts with error bars for confidence intervals for LCA with continuous, count variables are saved in the folder `_Plots` within subfolders.
`group.ind`	logical: if `TRUE` (default), latent class indicators are represented by separate bars when saving plots of an LCA based on continuous or count indicator variables, while latent class indicators are represented on the x-axis when saving plots of an LCA based on ordered or unordered categorical indicator, if `FALSE` latent classes are represented by separate bars when saving plots of an LCA based on continuous or count indicator variables, while latent classes are represented on the x-axis when saving plots of an LCA based on ordered or unordered categorical indicator.
`ci`	logical: if `TRUE` (default), confidence intervals are added to the bar charts for LCA with continuous or count indicator variables.
`axis.title`	a numeric value specifying the size of the axis title.
`axis.text`	a numeric value specifying the size of the axis text
`levels`	a character string specifying the order of the indicator variables shown on the x-axis.
`labels`	a character string specifying the labels of the indicator variables shown on the x-axis.
`ylim`	a numeric vector of length two specifying limits of the y-axis.
`ylab`	a character string specifying the label of the y-axis.
`breaks`	a numeric vector specifying the points at which tick-marks are drawn at the y-axis.
`errorbar.width`	a numeric vector specifying the width of the error bars. By default, the width of the error bars is 0.1 plus number of classes divided by 30.
`legend.title`	a numeric value specifying the size of the legend title.

legend.text — a numeric value specifying the size of the legend text.

legend.key.size — a numeric value specifying the size of the legend keys.

gray — logical: if TRUE, bar charts are drawn in gray scale.

start — a numeric value between 0 and 1 specifying the gray value at the low end of the palette.

end — a numeric value between 0 and 1 specifying the gray value at the high end of the palette.

dpi — a numeric value specifying the plot resolution when saving the bar chart.

width.ind — a numeric value specifying the width of the plot as a factor depending on the number of indicator variables. By default, the factor is 1.5.

width.nclass — a numeric value specifying the width of the plot as a factor depending on the number of classes. By default, the factor is 0.5 when saving plots of an LCA based on continuous or count indicator variables, while the factor is 1.5 when saving plots of an LCA based on ordered or unordered categorical indicator variables.

height.categ — a numeric value specifying the height of the plot as a factor depending on the number of response categories. By default, the factor is 0.6. Note that this argument is used only when saving plots of an LCA based on ordered or unordered categorical indicator variables.

height — a numeric value specifying the height of the plot when saving the bar chart. Note that this argument is used only when saving plots of an LCA based on continuous or count indicator variables.

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification is checked.

output — logical: if TRUE (default), output is shown.

Details

The Excel file exported by the function for reading Mplus output files from latent class analysis with continuous or count indicator variables by specifying the write argument (e.g., write = "Results_LCA.xlsx") contains five sheets.

(1) Summary: Model Summaries

- "Folder": Subfolder from which the group of Mplus outputs files were summarized
- "#Class": Number of latent classes, i.e., CLASSES ARE c(#Class)
- "Conv": Model converged, TRUE or FALSE, i.e., THE MODEL ESTIMATION TERMINATED NORMALLY
- "#Param": Number of estimated parameters, i.e., Number of Free Parameters
- "logLik": Log-likelihood of the estimated model, i.e., H0 Value

- "Scale": Scaling correction factor, i.e., H0 Scaling Correction Factor for, available only when ESTIMATOR IS MLR
- "LLRep": Best log-likelihood replicated, TRUE or FALSE, i.e., THE BEST LOGLIKELIHOOD VALUE HAS BEEN REPLICATED
- "AIC": Akaike information criterion, i.e., Akaike (AIC)
- "CAIC": Consistent AIC, not reported in the Mplus output, but simply BIC + #Param
- "BIC": Bayesian information criterion, i.e., Bayesian (BIC)
- "SABIC": Sample-size adjusted BIC, i.e., Sample-Size Adjusted BIC
- "AWE": Approximate weight of evidence criterion (Banfield & Raftery, 1993)
- "cmP": Approximate correct model probability (Schwarz, 1978) across estimated models in all Mplus output files in the subfolders to create result tables
- "Chi-Pear": Pearson chi-square test of model fit, i.e., Pearson Chi-Square, available only when indicators are count or ordered categorical
- "Chi-LRT": Likelihood ratio chi-square test of model fit, i.e., Likelihood Ratio Chi-Square, available only when indicators are count or ordered categorical
- "LMR-LRT": Significance value (p-value) of the Vuong-Lo-Mendell-Rubin test, i.e., VUONG-LO-MENDELL-RUBIN LIKELIHOOD RATIO TEST, available only when OUTPUT: TECH11
- "A-LRT": Significance value (p-value) of the Adjusted Lo-Mendell-Rubin Test, i.e., LO-MENDELL-RUBIN ADJUSTED LRT TEST, available only when OUTPUT: TECH11
- "BLRT": Significance value (p-value) of the bootstrapped likelihood ratio test, available only when OUTPUT: TECH14
- "Entropy": Summary of the class probabilities across classes and individuals in the sample, i.e., Entropy
- "aPPMin": Minimum average posterior class probability (AvePP) for the latent classes
- "OCCMin": Minimum odds of correct classification ratio (OCC)
- "nMin": Minimum class count for the latent classes based on the estimated model
- "pMin": Minimum class proportion for the latent classes based on the estimated model

(2) aBF: Approximate Bayes Factors

- "A-Folder": Subfolder from which the group of Mplus outputs files for Model A were summarized
- "A-#Class": Number of latent classes for Model A, i.e., CLASSES ARE c(#Class)
- "A-BIC": Bayesian information criterion for Model A, i.e., Bayesian (BIC)
- "B-Folder": Subfolder from which the group of Mplus outputs files for Model B were summarized
- "B-#Class": Number of latent classes for Model B, i.e., CLASSES ARE c(#Class)
- "B-BIC": Bayesian information criterion for Model B, i.e., Bayesian (BIC)
- "aBF": Approximate Bayes Factor for pairwise comparison of relative fit between Model A and Model B, i.e., ratio of the probability of Model A being correct model to Model B being the correct model

(3) Classif: Classification Diagnostics

- "Folder": Subfolder from which the group of Mplus outputs files were summarized
- "#Class": Number of latent classes, i.e., CLASSES ARE c(#Class).
- "Conv": Model converged, TRUE or FALSE, i.e., THE MODEL ESTIMATION TERMINATED NORMALLY.
- "#Param": Number of estimated parameters, i.e., Number of Free Parameters
- "LLRep": Best log-likelihood replicated, TRUE or FALSE, i.e., THE BEST LOGLIKELIHOOD VALUE HAS BEEN REPLICATED
- "n1": Class count for the first latent class based on the estimated model, i.e., FINAL CLASS COUNTS AND PROPORTIONS
- "n2": Class count for the second latent class based on the estimated model, i.e., FINAL CLASS COUNTS AND PROPORTIONS
- "p1": Class proportion of the first class based on the estimated posterior probabilities, i.e., FINAL CLASS COUNTS AND PROPORTIONS
- "p2": Class proportion of the second class based on the estimated posterior probabilities, i.e., FINAL CLASS COUNTS AND PROPORTIONS
- "Entropy": Summary of the class probabilities across classes and individuals in the sample, i.e., Entropy
- "aPP1": Average posterior class probability (AvePP) of the first latent class for the latent classes
- "aPP2": Average posterior class probability (AvePP) of the second latent class for the latent classes
- "OCC1": Odds of correct classification ratio (OCC) of the first latent class
- "OCC2": Odds of correct classification ratio (OCC) of the second latent class

(4) Mean_Var: Means and Variances for each Latent Class Separately

- "Folder": Subfolder from which the group of Mplus outputs files were summarized
- "#Class": Number of latent classes, i.e., CLASSES ARE c(#Class)
- "n": Class counts based on the estimated model, i.e., FINAL CLASS COUNTS AND PROPORTIONS
- "Param": Parameter, i.e., mean or variance
- "Ind": Latent class indicator variable
- "Est.": Parameter estimate.
- "SE": Standard error
- "z": Test statistic
- "pval": Significance value
- "Low": Lower bound of the confidence interval
- "Upp": Upper bound of the confidence interval

(5) d: Cohen's d

- "Folder": Subfolder from which the group of Mplus outputs files were summarized

- "#Class": Number of latent classes, i.e., CLASSES ARE c(#Class)
- "Ind": Latent class indicator variable
- "Class.j": Number of classes for model *j*
- "Class.k": Number of classes for model *k*
- "n.j": Latent classes *j*
- "M.j": Class-specific mean of the indicator for the latent class *j*
- "SD.j": Class-specific standard deviation of the indicator for the latent class *j*
- "n.k": Latent classes *k*
- "M.k": Class-specific mean of the indicator for the latent class *k*
- "SD.k": Class-specific standard deviation of the indicator for the latent class *k*
- "d": Cohen's d, Standardized mean difference

For more info on fit indices, classification diagnostics, and evaluating class separation see Masyn (2013) and Sorgente et al. (2025).

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
output	list with all Mplus outputs
args	specification of function arguments
result	list with result tables, i.e., summary for the model summaries, bf for approximate Bayes factors, classif classification diagnostics, mean_var for class-specific means and variances of the indicator variables, prob for class-specific item response probabilities of the indicator variables and d for Cohen's d standardized mean difference between latent class *j* and latent class *k*

Author(s)

Takuya Yanagida

References

Banfield, J. D., & Raftery, A E. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics, 49*, 803-821.

Masyn, K. E. (2013). Latent class analysis and finite mixture modeling. In T. D. Little (Ed.), *The Oxford handbook of quantitative methods: Statistical analysis* (pp. 551–611). Oxford University Press.

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

Schwartz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics, 6*, 461-464.

Sorgente, A., Caliciuri, R., Robba, M., Lanz, M., & Zumbo, B. D. (2025) A systematic review of latent class analysis in psychology: Examining the gap between guidelines and research practice. *Behavior Research Methods, 57*(11), 301. https://doi.org/10.3758/s13428-025-02812-1

See Also

mplus.lca, mplus.run, read.mplus, write.mplus

Examples

```
## Not run:

# Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

# Run LCA with k = 1 to k = 6 classes
mplus.lca(HolzingerSwineford1939, ind = c("x1", "x2", "x3", "x4"),
          mplus.run = TRUE)

#————————————————————————————————————————————————————————————————————
# Example 1: Summary Result Tables and Grouped Bar Charts

# Example 1a: Read Mplus output files, create result table, write table, and save plots
mplus.lca.summa(write = "Results_LCA.xlsx", plot = TRUE)

# Example 1b: Write results into a text file
mplus.lca.summa(write = "Results_LCA.txt")

#————————————————————————————————————————————————————————————————————
# Example 2: Draw bar chart manually

library(ggplot2)

# Collect LCA results
lca.result <- mplus.lca.summa()

# Result table with means
means <- lca.result$result$mean

# Extract results from variance-covariance structure A with 4 latent classes
plotdat <- means[means$folder == "A_Invariant-Theta_Diagonal-Sigma" &  means$nclass == 4, ]

# Draw bar chart
ggplot(plotdat, aes(ind, est, group = class, fill = class)) +
  geom_bar(stat = "identity", position = "dodge", color = "black",
           linewidth = 0.1) +
  geom_errorbar(aes(ymin = low, ymax = upp), width = 0.23,
                linewidth = 0.2, position = position_dodge(0.9)) +
  scale_x_discrete("") +
  scale_y_continuous("Mean Value", limits = c(0, 9),
                     breaks = seq(0, 9, by = 1)) +
  labs(fill = "Latent Class") +
  guides(fill = guide_legend(nrow = 1L)) +
  theme(axis.title = element_text(size = 11),
        axis.text = element_text(size = 11),
        legend.position = "bottom",
        legend.key.size = unit(0.5 , 'cm'),
```

```
        legend.title = element_text(size = 11),
        legend.text = element_text(size = 11),
        legend.box.spacing = unit(-9L, "pt"))

# Save bar chart
ggsave("LCA_4-Class.png", dpi = 600, width = 6, height = 4)

## End(Not run)
```

`mplus.plot` *Plot Mplus GH5 File*

Description

This function uses the `h5file` function in the **hdf5r** package to read a Mplus GH5 file that is requested by the command `PLOT: TYPE IS PLOT2` in Mplus to display trace plots, posterior distribution plots, autocorrelation plots, posterior predictive check plots based on the "bayesian_data" section, and the loop plot based on the "loop_data" section of the Mplus GH5 file. By default, the function displays trace plots if the "bayesian_data" section is available in the Mplus GH5 File. Otherwise, the function plots the loop plot if the "loop_data" section is available in the Mplus GH5 file.

Usage

```
mplus.plot(x, plot = c("none", "trace", "post", "auto", "ppc", "loop"),
           param = c("all", "on", "by", "with", "inter", "var", "r2", "new"),
           std = c("all", "none", "stdyx", "stdy", "std"), burnin = TRUE,
           point = c("all", "none", "m", "med", "map"),
           ci = c("none", "eti", "hdi"), chain = 1, conf.level = 0.95,
           hist = TRUE, density = TRUE, area = TRUE, alpha = 0.4,
           fill = "gray85", facet.nrow = NULL, facet.ncol = NULL,
           facet.scales = c("fixed", "free", "free_x", "free_y"),
           xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
           xbreaks = ggplot2::waiver(), ybreaks = ggplot2::waiver(),
           xexpand = ggplot2::waiver(), yexpand = ggplot2::waiver(),
           palette = "Set 2", binwidth = NULL, bins = NULL,
           density.col = "#0072B2", shape = 21,
           point.col = c("#CC79A7", "#D55E00", "#009E73"),
           linewidth = 0.6, linetype = "dashed", line.col = "black",
           bar.col = "black", bar.width = 0.8, plot.margin = NULL,
         legend.title.size = 10, legend.text.size = 10, legend.box.margin = NULL,
           saveplot = c("all", "none", "trace", "post", "auto", "ppc", "loop"),
           filename = "Mplus_Plot.pdf",
           file.plot = c("_TRACE", "_POST", "_AUTO", "_PPC", "_LOOP"),
           width = NA, height = NA, units = c("in", "cm", "mm", "px"),
           dpi = 600, check = TRUE)
```

Arguments

`x`	a character string indicating the name of the Mplus GH5 file (HDF5 format) with or without the file extension `.gh5`, e.g., `"Mplus_Plot.gh5"` or `"Mplus_Plot"`. Alternatively, a `misty.object` of type `mplus` can be specified, i.e., result object of the `mplus.plot()` function.
`plot`	a character string indicating the type of plot to display, i.e., `"none"` for not displaying any plot, `"trace"` (default) for displaying trace plots, `post` for displaying posterior distribution plots, `"auto"` for displaying autocorrelation plots, `"ppc"` for displaying posterior predictive check plots, and `"loop"` for displaying the loop plot. The default setting is `"trace"` if the "bayesian_data" section is available in the Mplus GH5 file. Otherwise, the default setting switches to `"loop"`.
`param`	character vector indicating which parameters to print for the trace plots, posterior distribution plots, and autocorrelation plots, i.e., `"all"` for all parameters, `"on"` (default), for regression slopes, `"by"` for factor loadings, `"with"` for covariances, `"inter"` for intercepts and thresholds, `"var"` for (residual) variances, `"r2"` for r-square, and `"new"` for parameters not in the analysis model specified in the `NEW` option. The default setting is `"on"` if regression slopes are available. Otherwise, the default setting switches to `"by"` and to `"with"` if factor loadings are not available.
`std`	a character vector indicating the standardized parameters to print for the trace plots, posterior distribution plots, and autocorrelation plots, i.e., `"all"` for all standardized parameters, `"none"` (default) for not printing any standardized parameters, `"stdyx"` for StdYX standardized parameters, `"stdy"` for StdY standardized parameters, and `"std"` for StdX standardized parameters.
`burnin`	logical: if `FALSE`, the first half of each chain is discarded as being part of the burnin phase when displaying trace plots. The default setting for `plot = "trace"` is `TRUE`. Note that the first half of each chain is always discarded when displaying posterior distribution plots (`plot = "post"`) regardless of the setting of the argument `burnin`.
`point`	a character vector indicating the point estimate(s) to be displayed in the posterior distribution plots, i.e., `"all"` for all point estimates, `"none"` for not displaying any point estimates, `"m"` for the posterior mean estimate, `"med"` (default) for the posterior median estimate, and `"map"` for the maximum a posteriori estimate.
`ci`	a character string indicating the type of credible interval to be displayed in the posterior distribution plots, i.e., `"none"` for not displaying any credible intervals, `"eti"` (default) for displaying the equal-tailed intervals and `"hdi"` for displaying the highest density interval.
`chain`	a numerical value indicating the chain to be used for the autocorrelation plots. By default, the first chain is used.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the credible interval (default is `0.95`).
`hist`	logical: if `TRUE` (default), histograms are drawn in the posterior probability plots.
`density`	logical: if `TRUE` (default), density curves are drawn in the posterior probability plots.

area — logical: if TRUE (default), statistical not significant and statistical significant area is filled with a different color and vertical lines are drawn.

alpha — a numeric value between 0 and 1 for the alpha argument (default is 0.4) for the annotate, geom_histogram, geom_bar, and geom_ribbon function.

fill — a character string indicating the color for the "fill" argument (default is "gray85") for the annotate, geom_histogram, geom_bar, and geom_point functions.

facet.nrow — a numeric value indicating the nrow argument (default is NULL) for the facet_wrap function.

facet.ncol — a numeric value indicating the ncol argument (default is 2) for the facet_wrap function.

facet.scales — a character string indicating the scales argument (default is "free") for the facet_wrap function.

xlab — a character string indicating the name argument for the scale_x_continuous function. Note that the default setting depends on the type of plot, e.g., "" for the trace plots and "Lag" for the autocorrelation plots.

ylab — a character string indicating the name argument for the scale_y_continuous function. Note that the default setting depends on the type of plot, e.g., "" for the trace plots and "Autocorrelation" for the autocorrelation plots.

xlim — a numeric vector with two elements indicating the limits argument (default it NULL) for the scale_x_continuous function.

ylim — a numeric vector with two elements indicating the limits argument (default it NULL) for the scale_y_continuous function.

xbreaks — a numeric vector indicating the breaks argument (default is ggplot2::waiver()) for the scale_x_continuous function.

ybreaks — a numeric vector indicating the breaks argument (default is ggplot2::waiver()) for the scale_y_continuous function.

xexpand — a numeric vector with two elements indicating the expand argument (default is (0.02, 0)) for the scale_x_continuous function.

yexpand — a numeric vector with two elements indicating the expand argument for the scale_y_continuous function. Note that the default setting depends on the type of plot, e.g., (0.02, 0) for the trace plots and expansion(mult = c(0, 0.05)) for the posterior distribution plots.

palette — a character string indicating the palette name (default is "Set 2") for the hcl.colors function. Note that the character string must be one of hcl.pals().

binwidth — a numeric value indicating the binwidth argument (default is to use the number of bins in bins argument) for the geom_histogram function.

bins — a numeric value indicating the bins argument (default is 30) for the geom_histogram function.

density.col — a character string indicating the color argument (default is "#0072B2") for the geom_density function.

shape — a numeric value indicating the shape argument (default is 21) for the geom_point function.

point.col
: a character vector with three elements indicating the values argument (default is c("#CC79A7", "#D55E00", "#009E73")) for the scale_color_manual function.

linewidth
: a numeric value indicating the linewidth argument (default is 0.6) for the geom_vline function.

linetype
: a numeric value indicating the linetype argument (default is "dashed") for the geom_vline function.

line.col
: a character string indicating the color argument (default is "black") for the geom_vline function.

bar.col
: a character string indicating the color argument (default is "black") for the geom_bar function.

bar.width
: a character string indicating the width argument (default = 0.8)for the geom_bar function.

plot.margin
: a numeric vector indicating the plot.margin argument for the theme function. Note that the default setting depends on the type of the plot, e.g., c(4, 15, -10, 0) for the trace plots, and c(4, 15, 4, 4) for the autocorrelation plots.

legend.title.size
: a numeric value indicating the legend.title argument (default is element_text(size = 10)) for the theme function.

legend.text.size
: a numeric value indicating the legend.text argument (default is element_text(size = 10)) for the theme function.

legend.box.margin
: a numeric vector indicating the legend.box.margin argument for the theme function. Note that the default setting depends on the type of plot, e.g., c(-16, 6, 6, 6) for the trace plots, and c(-25, 6, 6, 6) for the posterior distribution plots with displaying point estimates.

saveplot
: a character vector indicating the plot to be saved, i.e., "all" for saving all plots, "none" (default) for not saving any plots, "trace" for saving the trace plots, post for the saving the posterior distribution plots, "auto" for saving the autocorrelation plots, "ppc" for saving the posterior predictive check plots, and "loop" for saving the loop plot.

filename
: a character string indicating the filename argument (default is "Mplus_Plot.pdf") including the file extension for the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument.

file.plot
: a character vector with five elements for distinguishing different types of plots. By default, the character string specified in the argument "filename" ("Mplus_Plot") is concatenated with "_TRACE" ("Mplus_Plot_TRACE") for the trace plots, "_POST" ("Mplus_Plot_POST") for the posterior distribution plots, "_AUTO" ("Mplus_Plot_AUTO") for the autocorrelation plots, "_PPC" ("Mplus_Plot_PPC") for the posterior predictive check plots, and "_LOOP" ("Mplus_Plot_LOOP") for the loop plot.

width
: a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.

height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
units	a character string indicating the units argument (default is in) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	Mplus GH5 file
args	specification of function arguments
data	list with posterior distribution of each parameter estimate in wide and long format (post), autocorrelation for each parameter estimate in wide and long format (auto), data for the posterior predictive check (ppc), and data for the loop plot (loop)
plot	list with the trace plots (trace, posterior distribution plots (post), autocorrelation plots (auto), posterior predictive check plots (ppc), and loop plot (loop)

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.mplus, write.mplus, mplus, mplus.update, mplus.print, mplus.bayes, mplus.run, mplus.lca

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————
# Mplus Example 3.18: Moderated Mediation with a Plot of the Indirect Effect

#————————————————————————————————————
## Trace Plots

# Example 1a: Default setting
mplus.plot("ex3.18.gh5")
```

```
# Example 1b: Exclude first half of each chain
mplus.plot("ex3.18.gh5", burnin = FALSE)

# Example 1c: Print all parameters
mplus.plot("ex3.18.gh5", param = "all")

# Example 1d: Print user-specified parameters
mplus.plot("ex3.18.gh5", param = "param")

# Example 1e: Arrange panels in three columns
mplus.plot("ex3.18.gh5", ncol = 3)

# Example 1f: Specify "Pastel 1" palette for the hcl.colors function
mplus.plot("ex3.18.gh5", palette = "Pastel 1")

#————————————————————————————————————
## Posterior Distribution Plots

# Example 2a: Default setting, i.e., posterior median and equal-tailed interval
mplus.plot("ex3.18.gh5", plot = "post")

# Example 2b: Display posterior mean and maximum a posteriori
mplus.plot("ex3.18.gh5", plot = "post", point = c("m", "map"))

# Example 2c: Display maximum a posteriori and highest density interval
mplus.plot("ex3.18.gh5", plot = "post", point = "map", ci = "hdi")

# Example 2d: Do not display any point estimates and credible interval
mplus.plot("ex3.18.gh5", plot = "post", point = "none", ci = "none")

# Example 2d: Do not display histograms
mplus.plot("ex3.18.gh5", plot = "post", hist = FALSE)

#————————————————————————————————————
## Autocorrelation Plots

# Example 3a: Default setting, i.e., first chain
mplus.plot("ex3.18.gh5", plot = "auto")

# Example 3b: Use second chain
mplus.plot("ex3.18.gh5", plot = "auto", chain = 2)

# Example 3b: Modify limits and breaks of the y-axis
mplus.plot("ex3.18.gh5", plot = "auto",
           ylim = c(-0.05, 0.05), ybreaks = seq(-0.1, 0.1, by = 0.025))

#————————————————————————————————————
## Posterior Predictive Check Plots

# Example 4a: Default setting, i.e., 95% Interval
mplus.plot("ex3.18.gh5", plot = "ppc")

# Example 4b: Default setting, i.e., 99% Interval
```

```
mplus.plot("ex3.18.gh5", plot = "ppc", conf.level = 0.99)

#————————————————————————————————————
## Loop Plot

# Example 5a: Default setting
mplus.plot("ex3.18.gh5", plot = "loop")

# Example 5b: Do not fill area and draw vertical lines
mplus.plot("ex3.18.gh5", plot = "loop", area = FALSE)

#————————————————————————————————————
## Save Plots

# Example 6a: Save all plots in pdf format
mplus.plot("ex3.18.gh5", saveplot = "all")

# Example 6b: Save all plots in png format with 300 dpi
mplus.plot("ex3.18.gh5", saveplot = "all", filename = "Mplus_Plot.png", dpi = 300)

# Example 6a: Save loop plot, specify width and height of the plot
mplus.plot("ex3.18.gh5", plot = "none", saveplot = "loop",
           width = 7.5, height = 7)

#————————————————————————————————————————————————————————————————————————
# Plot from misty.object

# Create misty.object
object <- mplus.plot("ex3.18.gh5", plot = "none")

# Trace plot
mplus.plot(object, plot = "trace")

# Posterior distribution plot
mplus.plot(object, plot = "post")

# Autocorrelation plot
mplus.plot(object, plot = "auto")

# Posterior predictive check plot
mplus.plot(object, plot = "ppc")

# Loop plot
mplus.plot(object, plot = "loop")

#————————————————————————————————————————————————————————————————————————
# Create Plots Manually

# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- mplus.plot("ex3.18.gh5", plot = "none")
```

```
#————————————————————————————————————
## Example 7: Trace Plots

# Extract data in long format
data.post <- object$data$post$long

# Extract ON parameters
data.trace <- data.post[grep(" ON ", data.post$param), ]

# Plot
ggplot(data.trace, aes(x = iter, y = value, color = chain)) +
  annotate("rect", xmin = 0, xmax = 15000, ymin = -Inf, ymax = Inf,
           alpha = 0.4, fill = "gray85") +
  geom_line() +
  facet_wrap(~ param, ncol = 2, scales = "free") +
  scale_x_continuous(name = "", expand = c(0.02, 0)) +
  scale_y_continuous(name = "", expand = c(0.02, 0)) +
  scale_colour_manual(name = "Chain",
                      values = hcl.colors(n = 2, palette = "Set 2")) +
  theme_bw() +
  guides(color = guide_legend(nrow = 1, byrow = TRUE)) +
  theme(plot.margin = margin(c(4, 15, -10, 0)),
        legend.position = "bottom",
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 10),
        legend.box.margin = margin(c(-16, 6, 6, 6)),
        legend.background = element_rect(fill = "transparent"))

#————————————————————————————————————
## Example 8: Posterior Distribution Plots

# Extract data in long format
data.post <- object$data$post$long

# Extract ON parameters
data.post <- data.post[grep(" ON ", data.post$param), ]

# Discard burn-in iterations
data.post <- data.post[data.post$iter > 15000, ]

# Drop factor levels
data.post$param <- droplevels(data.post$param,
                              exclude = c("[Y]", "[M]", "Y", "M", "INDIRECT", "MOD"))

# Plot
ggplot(data.post, aes(x = value)) +
  geom_histogram(aes(y = after_stat(density)), color = "black", alpha = 0.4,
                 fill = "gray85") +
  geom_density(color = "#0072B2") +
  geom_vline(data = data.frame(param = unique(data.post$param),
                               stat = tapply(data.post$value, data.post$param, median)),
             aes(xintercept = stat, color = "Median"), linewidth = 0.6) +
```

```
  geom_vline(data = data.frame(param = unique(data.post$param),
                               low = tapply(data.post$value, data.post$param,
                                            function(y) quantile(y, probs = 0.025))),
             aes(xintercept = low), linetype = "dashed", linewidth = 0.6) +
  geom_vline(data = data.frame(param = unique(data.post$param),
                               upp = tapply(data.post$value, data.post$param,
                                            function(y) quantile(y, probs = 0.975))),
             aes(xintercept = upp), linetype = "dashed", linewidth = 0.6) +
  facet_wrap(~ param, ncol = 2, scales = "free") +
  scale_x_continuous(name = "", expand = c(0.02, 0)) +
  scale_y_continuous(name = "Probability Density, f(x)",
                     expand = expansion(mult = c(0L, 0.05))) +
  scale_color_manual(name = "Point Estimate", values = c(Median = "#D55E00")) +
  labs(caption = "95% Equal-Tailed Interval") +
  theme_bw() +
  theme(plot.margin = margin(c(4, 15, -8, 4)),
        plot.caption = element_text(hjust = 0.5, vjust = 7),
        legend.position = "bottom",
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 10),
        legend.box.margin = margin(c(-30, 6, 6, 6)),
        legend.background = element_rect(fill = "transparent"))

#————————————————————————————————————
## Example 9: Autocorrelation Plots

# Extract data in long format
data.auto <- object$data$auto$long

# Select first chain
data.auto <- data.auto[data.auto$chain == 1, ]

# Extract ON parameters
data.auto <- data.auto[grep(" ON ", data.auto$param), ]

# Plot
ggplot(data.auto, aes(x = lag, y = cor)) +
  geom_bar(stat = "identity", alpha = 0.4, color = "black", fill = "gray85",
           width = 0.8) +
  facet_wrap(~ param, ncol = 2) +
  scale_x_continuous(name = "Lag", breaks = seq(1, 30, by = 2), expand = c(0.02, 0)) +
  scale_y_continuous(name = "Autocorrelation", limits = c(-0.1, 0.1),
                     breaks = seq(-0.1, 1., by = 0.05), expand = c(0.02, 0)) +
  theme_bw() +
  theme(plot.margin = margin(c(4, 15, 4, 4)))

#————————————————————————————————————
## Example 10: Posterior Predictive Check (PPC) Plots

# Extract data
data.ppc <- object$data$ppc

# Scatter plot
```

```
ppc.scatter <- ggplot(data.ppc, aes(x = obs, y = rep)) +
  geom_point(shape = 21, fill = "gray85") +
  geom_abline(slope = 1) +
  scale_x_continuous("Observed", limits = c(0, 45), breaks = seq(0, 45, by = 5),
                     expand = c(0.02, 0)) +
  scale_y_continuous("Recpliated", limits = c(0, 45), breaks = seq(0, 45, by = 5),
                     expand = c(0.02, 0)) +
  theme_bw() +
  theme(plot.margin = margin(c(2, 15, 4, 4)))

# Histogram
ppc.hist <- ggplot(data.ppc, aes(x = diff)) +
  geom_histogram(color = "black", alpha = 0.4, fill = "gray85") +
  geom_vline(xintercept = mean(data.ppc$diff), color = "#CC79A7") +
  geom_vline(xintercept = quantile(data.ppc$diff, probs = 0.025),
             linetype = "dashed", color = "#CC79A7") +
  geom_vline(xintercept = quantile(data.ppc$diff, probs = 0.975),
             linetype = "dashed", color = "#CC79A7") +
  scale_x_continuous("Observed - Replicated", expand = c(0.02, 0)) +
  scale_y_continuous("Count", expand = expansion(mult = c(0L, 0.05))) +
  theme_bw() +
  theme(plot.margin = margin(c(2, 15, 4, 4)))

# Combine plots using the patchwork package
patchwork::wrap_plots(ppc.scatter, ppc.hist)

#————————————————————————————————————
## Example 11: Loop Plot

# Extract data
data.loop <- object$data$loop

# Plot
plot.loop <- ggplot(data.loop, aes(x = xval, y = estimate)) +
  geom_line(linewidth = 0.6, show.legend = FALSE) +
  geom_line(aes(xval, low)) +
  geom_line(aes(xval, upp)) +
  scale_x_continuous("MOD", expand = c(0.02, 0)) +
  scale_y_continuous("INDIRECT", expand = c(0.02, 0)) +
  scale_fill_manual("Statistical Significance",
                    values = hcl.colors(n = 2, palette = "Set 2")) +
  theme_bw() +
  theme(plot.margin = margin(c(4, 15, -6, 4)),
        legend.position = "bottom",
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 10),
        legend.box.margin = margin(-10, 6, 6, 6),
        legend.background = element_rect(fill = "transparent"))

# Significance area
for (i in unique(data.loop$group)) {

  plot.loop <- plot.loop + geom_ribbon(data = data.loop[data.loop$group == i, ],
```

```
                                    aes(ymin = low, ymax = upp, fill = sig), alpha = 0.4)

}

# Vertical lines
plot.loop + geom_vline(data = data.loop[data.loop$change == 1, ],
                       aes(xintercept = xval, color = sig), linewidth = 0.6,
                           linetype = "dashed", show.legend = FALSE)

## End(Not run)
```

`mplus.print` *Print Mplus Output*

Description

This function prints the input command sections and the result sections of a Mplus output file (`.out`) on the R console. By default, the function prints selected result sections, e.g., short `Summary of Analysis`, short `Summary of Data`, `Model Fit Information`, and `Model Results`.

Usage

```
mplus.print(x, print = c("all", "input", "result"),
            input = c("all", "default", "data", "variable", "define",
                      "analysis", "model", "montecarlo", "mod.pop", "mod.cov",
                      "mod.miss", "message"),
             result = c("all", "default", "summary.analysis.short",
                        "summary.data.short", "random.starts", "summary.fit",
                        "mod.est", "fit", "class.count", "classif",
                        "mod.result", "total.indirect"),
           exclude = NULL, variable = FALSE, not.input = TRUE, not.result = TRUE,
             write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`x`	a character string indicating the name of the Mplus output file with or without the file extension `.out`, e.g., `"Mplus_Output.out"` or `"Mplus_Output"`. Alternatively, a `misty.object` of type `mplus` can be specified, i.e., result object of the `mplus.print()`, `mplus()` or `mplus.update()` function.
`print`	a character vector indicating which section to show, i.e. `"all"` for input and result sections, `"input"` for input command section only, and `"result"` (default) for result sections only
`input`	a character vector specifying Mplus input command sections
`result`	a character vector specifying Mplus result sections included in the output (see 'Details').
`exclude`	a character vector specifying Mplus input command or result sections excluded from the output (see 'Details').

variable	logical: if TRUE, names of the variables in the data set (NAMES option) specified in the VARIABLE: command section are shown. By default, names of the variables in the data set are excluded from the output unless all variables are used in the analysis (i.e., no USEVARIABLES option specified in the Mplus input file).
not.input	logical: if TRUE (default), character vector indicating the input commands not requested are shown on the console.
not.result	logical: if TRUE (default), character vector indicating the result sections not requested are shown on the console.
write	a character string naming a file for writing the output into a text file with file extension ".txt" (e.g., "Output.txt").
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Input Command Sections Following input command sections can be selected by using the input argument or excluded by using the exclude argument:

- "title" for the TITLE command used to provide a title for the analysis.
- "data" for the DATA command used to provide information about the data set to be analyzed.
- "data.imp" for the DATA IMPUTATION command used to create a set of imputed data sets using multiple imputation methodology.
- "data.wl" for the DATA WIDETOLONG command used to rearrange data from a multivariate wide format to a univariate long format.
- "data.lw" for the DATA LONGTOWIDE command used to rearrange a univariate long format to a multivariate wide format.
- "data.tp" for the DATA TWOPART command used to create a binary and a continuous variable from a continuous variable with a floor effect for use in two-part modeling.
- "data.miss" for the DATA MISSING command used to create a set of binary variables that are indicators of missing data or dropout for another set of variables.
- "data.surv" for the DATA SURVIVAL command used to create variables for discrete-time survival modeling.
- "data.coh" for the DATA COHORT command used to rearrange longitudinal data from a format where time points represent measurement occasions to a format where time points represent age or another time-related variable,
- "variable" for the VARIABLE command used to provide information about the variables in the data set to be analyzed.
- "define" for the DEFINE command used to transform existing variables and to create new variables.
- "analysis" for the ANALYSIS command used to describe the technical details for the analysis.
- "model" MODEL for the command used to describe the model to be estimated.

- "mod.ind" for the MODEL INDIRECT command used to request indirect and directed effects and their standard errors.
- "mod.test" for the MODEL TEST command used to test restrictions on the parameters in the MODEL and MODEL CONSTRAINT commands using the Wald chi-square test.
- "mod.prior" for the MODEL PRIORS command used with ESTIMATOR IS BAYES to specify the prior distribution for each parameter.
- "montecarlo" for the MONTECARLO command used to set up and carry out a Monte Carlo simulation study.
- "mod.pop" for the MODEL POPULATION command used to provide the population parameter values to be used in data generation using the options of the MODEL command.
- "mod.cov" for the MODEL COVERAGE used to provide the population parameter values to be used for computing coverage.
- "mod.miss" for the MODEL MISSING command used to provide information about the population parameter values for the missing data model to be used in the generation of data.
- "output" for the for the OUTPUT command used to request additional output beyond that included as the default.
- "savedata" for the SAVEDATA command used to save the analysis data and/or a variety of model results in an ASCII file for future use.
- "plot" for the PLOT command used to requested graphical displays of observed data and analysis results.
- "message" for warning and error messages that have been generated by the program after the input command sections.

Note that all input command sections are requested by specifying input = "all". The input argument is also used to select one (e.g., input = "model") or more than one input command sections (e.g., input = c("analysis", "model")), or to request input command sections in addition to the default setting (e.g., input = c("default", "output")). The exclude argument is used to exclude input command sections from the output (e.g., exclude = "variable").

Result Sections Following result sections can be selected by using the result argument or excluded by using the exclude argument:

- "summary.analysis" for the SUMMARY OF ANALYSIS section..
- "summary.analysis.short" for a short SUMMARY OF ANALYSIS section including the number of observations, number of groups, estimator, and optimization algorithm.
- "summary.data" for the SUMMARY OF DATA section indicating.
- "summary.data.short" for a short SUMMARY OF DATA section including number of clusters, average cluster size, and estimated intraclass correlations.
- "prop.count" for the UNIVARIATE PROPORTIONS AND COUNTS FOR CATEGORICAL VARIABLES section.
- "summary.censor" for the SUMMARY OF CENSORED LIMITS section.
- "prop.zero" for the COUNT PROPORTION OF ZERO, MINIMUM AND MAXIMUM VALUES section.
- "crosstab" for the CROSSTABS FOR CATEGORICAL VARIABLES section.
- "summary.miss" for the SUMMARY OF MISSING DATA PATTERNS section.
- "coverage" for the COVARIANCE COVERAGE OF DATA section.
- "basic" for the RESULTS FOR BASIC ANALYSIS section.

- "sample.stat" for the SAMPLE STATISTICS section.
- "uni.sample.stat" for the UNIVARIATE SAMPLE STATISTICS section.
- "random.starts" for the RANDOM STARTS RESULTS section.
- "summary.fit" for the SUMMARY OF MODEL FIT INFORMATION section.
- "mod.est" for the THE MODEL ESTIMATION TERMINATED NORMALLY message and warning messages from the model estimation.
- "fit" for the MODEL FIT INFORMATION section.
- "class.count" for the FINAL CLASS COUNTS AND PROPORTIONS FOR THE LATENT CLASSES section.
- "ind.means" for the LATENT CLASS INDICATOR MEANS AND PROBABILITIES FOR EACH LATENT CLASS section.
- "equal.mean" for the EQUALITY TESTS OF MEANS ACROSS THE LATENT CLASSES section.
- "trans.prob" for the LATENT TRANSITION PROBABILITIES BASED ON THE ESTIMATED MODEL section.
- "classif" for the CLASSIFICATION QUALITY section.
- "mod.result" for the MODEL RESULTS and RESULTS FOR EXPLORATORY FACTOR ANALYSIS section.
- "odds.ratio" for the LOGISTIC REGRESSION ODDS RATIO RESULTS section.
- "odds.ratio.class" for the LOGISTIC REGRESSION ODDS RATIO RESULTS FOR LATENT CLASS VARIABLES section.
- "prob.scale" for the RESULTS IN PROBABILITY SCALE section.
- "multinom" for the CATEGORICAL LATENT VARIABLE MULTINOMIAL LOGISTIC REGRESSIONS USING THE 3-STEP (FIXED LOGITS) PROCEDURE section.
- "odds.ratio.multinom" for the ODDS RATIOS FOR CATEGORICAL LATENT VARIABLE MULTINOMIAL LOGISTIC REGRESSIONS USING THE 3-STEP (FIXED LOGITS) PROCEDURE section.
- "ind.odds.ratio" for the LATENT CLASS INDICATOR ODDS RATIOS FOR THE LATENT CLASSES section.
- "alt.param" for the ALTERNATIVE PARAMETERIZATIONS FOR THE CATEGORICAL LATENT VARIABLE REGRESSION section.
- "alt.param.odds.ratio" for the ODDS RATIOS FOR THE ALTERNATIVE PARAMETERIZATIONS FOR THE CATEGORICAL LATENT VARIABLE REGRESSIO section.
- "irt.param" for the IRT PARAMETERIZATION section.
- "equal.mean.bch" for the EQUALITY TESTS OF MEANS/PROBABILITIES ACROSS CLASSES section.
- "brant.wald" for the BRANT WALD TEST FOR PROPORTIONAL ODDS section.
- "std.mod.result" for the STANDARDIZED MODEL RESULTS section.
- "rsquare" for the R-SQUARE section.
- "total.indirect" for the TOTAL, TOTAL INDIRECT, SPECIFIC INDIRECT, AND DIRECT EFFECTS section.
- "std.total.indirect" for the STANDARDIZED TOTAL, TOTAL INDIRECT, SPECIFIC INDIRECT, AND DIRECT EFFECTS section.
- "std.mod.result.cluster" for the WITHIN-LEVEL STANDARDIZED MODEL RESULTS FOR CLUSTER section.
- "fs.comparison" for the BETWEEN-LEVEL FACTOR SCORE COMPARISONS section.

- "conf.mod.result" for the CONFIDENCE INTERVALS OF MODEL RESULTS section.
- "conf.std.conf" for the CONFIDENCE INTERVALS OF STANDARDIZED MODEL RESULTS section.
- "conf.total.indirect" for the CONFIDENCE INTERVALS OF TOTAL, TOTAL INDIRECT, SPECIFIC INDIRECT, AND DIRECT EFFECTS section.
- "conf.odds.ratio" for the CONFIDENCE INTERVALS FOR THE LOGISTIC REGRESSION ODDS RATIO RESULTS section.
- "modind" for the MODEL MODIFICATION INDICES section.
- "resid" for the RESIDUAL OUTPUT section.
- "logrank" for the LOGRANK OUTPUT section.
- "tech1" for the TECHNICAL 1 OUTPUT section.
- "tech2" for the TECHNICAL 2 OUTPUT section.
- "tech3" for the TECHNICAL 3 OUTPUT section.
- "h1.tech3" for the H1 TECHNICAL 3 OUTPUT section.
- "tech4" for the TECHNICAL 4 OUTPUT section.
- "tech5" for the TECHNICAL 5 OUTPUT section.
- "tech6" for the TECHNICAL 6 OUTPUT section.
- "tech7" for the TECHNICAL 7 OUTPUT section.
- "tech8" for the TECHNICAL 8 OUTPUT section.
- "tech9" for the TECHNICAL 9 OUTPUT section.
- "tech10" for the TECHNICAL 10 OUTPUT section.
- "tech11" for the TECHNICAL 11 OUTPUT section.
- "tech12" for the TECHNICAL 12 OUTPUT section.
- "tech13" for the TECHNICAL 13 OUTPUT section.
- "tech14" for the TECHNICAL 14 OUTPUT section.
- "tech15" for the TECHNICAL 15 OUTPUT section.
- "tech16" for the TECHNICAL 16 OUTPUT section.
- "svalues" for the MODEL COMMAND WITH FINAL ESTIMATES USED AS STARTING VALUES section.
- "stat.fscores" for the SAMPLE STATISTICS FOR ESTIMATED FACTOR SCORES section.
- "summary.fscores" for the SUMMARY OF FACTOR SCORES section.
- "pv" for the SUMMARIES OF PLAUSIBLE VALUES section.
- "plotinfo" for the PLOT INFORMATION section.
- "saveinfo" for the SAVEDATA INFORMATION section.

Note that all result sections are requested by specifying result = "all". The result argument is also used to select one (e.g., result = "mod.result") or more than one result sections (e.g., result = c("mod.result", "std.mod.result")), or to request result sections in addition to the default setting (e.g., result = c("default", "odds.ratio")). The exclude argument is used to exclude result sections from the output (e.g., exclude = "mod.result").

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	character string or misty object
args	specification of function arguments
print	print objects
notprint	character vectors indicating the input commands and result sections not requested
result	list with input command sections (input) and result sections (input)

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.mplus, write.mplus, mplus, mplus.update, mplus.plot, mplus.bayes, mplus.run, mplus.lca

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————————
# Mplus Example 3.1: Linear Regression

# Example 1a: Default setting
mplus.print("ex3.1.out")

# Example 1b:  Print result section only
mplus.print("ex3.1.out", print = "result")

# Example 1c: Print MODEL RESULTS only
mplus.print("ex3.1.out", print = "result", result = "mod.result")

# Example 1d: Print UNIVARIATE SAMPLE STATISTICS in addition to the default setting
mplus.print("ex3.1.out", result = c("default", "uni.sample.stat"))

# Example 1e: Exclude MODEL FIT INFORMATION section
mplus.print("ex3.1.out", exclude = "fit")

# Example 1f: Print all result sections, but exclude MODEL FIT INFORMATION section
mplus.print("ex3.1.out", result = "all", exclude = "fit")

# Example 1g: Print result section in a different order
mplus.print("ex3.1.out", result = c("mod.result", "fit", "summary.analysis"))

#————————————————————————————————————————————————————————————————————————
```

```
# misty.object of type 'mplus.print'

# Example 2
# Create misty.object
object <- mplus.print("ex3.1.out", output = FALSE)

# Print misty.object
mplus.print(object)

#------------------------------------------------------------------------
# Write Results

# Example 3: Write Results into a text file
mplus.print("ex3.1.out", write = "Output_3-1.txt")

## End(Not run)
```

`mplus.run` *Run Mplus Models*

Description

This function runs a group of Mplus models (`.inp` files) located within a single directory or nested within subdirectories.

Usage

```
mplus.run(target = getwd(), recursive = FALSE, filefilter = NULL, show.out = FALSE,
          replace.out = c("always", "never", "modified"), message = TRUE,
          logFile = NULL, Mplus = .detect.mplus(), killOnFail = TRUE,
          local_tmpdir = FALSE, check = TRUE)
```

Arguments

`target`	a character string indicating the directory containing Mplus input files (`.inp`) to run or the single `.inp` file to be run. May be a full path, relative path, or a filename within the working directory.
`recursive`	logical: if `TRUE`, run all models nested in subdirectories within directory. Not relevant if target is a single file.
`filefilter`	a Perl regular expression (PCRE-compatible) specifying particular input files to be run within directory. See regex or http://www.pcre.org/pcre.txt for details about regular expression syntax. Not relevant if target is a single file.
`show.out`	logical: if `TRUE`, estimation output (`TECH8`) is show on the R console. Note that if run within Rgui, output will display within R, but if run via Rterm, a separate window will appear during estimation.

`replace.out`	a character string for specifying three settings: `"always"` (default), which runs all models, regardless of whether an output file for the model exists, `"never"`, which does not run any model that has an existing output file, and `"modified"`, which only runs a model if the modified date for the input file is more recent than the output file modified date.
`message`	logical: if `TRUE`, message `Running model:` and `System command:` is printed on the console.
`logFile`	a character string specifying a file that records the settings passed into the function and the models run (or skipped) during the run.
`Mplus`	a character string for specifying the name or path of the Mplus executable to be used for running models. This covers situations where Mplus is not in the system's path, or where one wants to test different versions of the Mplus program. Note that there is no need to specify this argument for most users since it has intelligent defaults.
`killOnFail`	logical: if `TRUE` (default), all processes named mplus.exe when `mplus.run()` does not terminate normally are killed. Windows only.
`local_tmpdir`	logical: if `TRUE`, the TMPDIR environment variable is set to the location of the `.inp file` prior to execution. This is useful in Monte Carlo studies where many instances of Mplus may run in parallel and we wish to avoid collisions in temporary files among processes. Linux/Mac only.
`check`	logical: if `TRUE` (default), argument specification, convergence and model identification is checked.

Value

None.

Note

This function is a copy of the `runModels()` function in the **MplusAutomation** package by Michael Hallquist and Joshua Wiley (2018).

Author(s)

Takuya Yanagida

References

Hallquist, M. N. & Wiley, J. F. (2018). MplusAutomation: An R package for facilitating large-scale latent variable analyses in Mplus. *Structural Equation Modeling: A Multidisciplinary Journal, 25*, 621-638. https://doi.org/10.1080/10705511.2017.1402334.

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

`read.mplus`, `write.mplus`, `mplus`, `mplus.update`, `mplus.print`, `mplus.plot`, `mplus.bayes`, `mplus.lca`

Examples

```
## Not run:

# Example 1: Run Mplus models located within a single directory
mplus.run(Mplus = "C:/Program Files/Mplus/Mplus.exe")

# Example 2: Run Mplus models located nested within subdirectories
mplus.run(recursive = TRUE,
          Mplus = "C:/Program Files/Mplus/Mplus.exe")

## End(Not run)
```

`mplus.update` *Mplus Input Updating*

Description

This function updates specific input command sections of a `misty.object` of type `mplus` to create an updated Mplus input file, run the updated input file by using the `mplus.run()` function, and print the updated Mplus output file by using the `mplus.print()` function.

Usage

```
mplus.update(x, update, file = "Mplus_Input_Update.inp", comment = FALSE,
             replace.inp = TRUE, mplus.run = TRUE,
             show.out = FALSE, replace.out = c("always", "never", "modified"),
             print = c("all", "input", "result"),
             input = c("all", "default", "data", "variable", "define",
                       "analysis", "model", "montecarlo", "mod.pop", "mod.cov",
                       "mod.miss", "message"),
             result = c("all", "default", "summary.analysis.short",
                        "summary.data.short", "random.starts", "summary.fit",
                        "mod.est", "fit", "class.count", "classif",
                        "mod.result", "total.indirect"),
           exclude = NULL, variable = FALSE, not.input = TRUE, not.result = TRUE,
             write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`x`	`misty.object` object of type `mplus`.
`update`	a character string containing the updated input command sections.
`file`	a character string indicating the name of the updated Mplus input file with or without the file extension `.inp`, e.g., `"Mplus_Input_Update.inp"` or `"Mplus_Input_Update"`.
`comment`	logical: if `FALSE` (default), comments (i.e., text after the `!` symbol) are removed from the input text specified in the argument `x`.
`replace.inp`	logical: if `TRUE` (default), an existing input file will be replaced.

mplus.run	logical: if TRUE, the input file specified in the argument file containing the input text specified in the argument x is run using the mplus.run function.
show.out	logical: if TRUE, estimation output (TECH8) is show on the R console. Note that if run within Rgui, output will display within R, but if run via Rterm, a separate window will appear during estimation.
replace.out	a character string for specifying three settings: "always" (default), which runs all models, regardless of whether an output file for the model exists, "never", which does not run any model that has an existing output file, and "modified", which only runs a model if the modified date for the input file is more recent than the output file modified date.
print	a character string indicating which results to show, i.e. "all" (default) for all results "input" for input command sections, and "result" for result sections.
input	a character vector specifying Mplus input command sections included in the output (see 'Details' in the mplus.print function).
result	a character vector specifying Mplus result sections included in the output (see 'Details' in the mplus.print function).
exclude	a character vector specifying Mplus input command or result sections excluded from the output (see 'Details' in the mplus.print function).
variable	logical: if TRUE, names of the variables in the data set (NAMES ARE) specified in the VARIABLE: command section are shown. By default, names of the variables in the data set are excluded from the output unless all variables are used in the analysis (i.e., no USEVARIABLES command specified in the Mplus input file).
not.input	logical: if TRUE (default), character vector indicating the input commands not requested are shown on the console.
not.result	logical: if TRUE (default), character vector indicating the result sections not requested are shown on the console.
write	a character string naming a file for writing the output into a text file with file extension ".txt" (e.g., "Output.txt").
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console by using the function mplus.print.

Details

Mplus Input Sections The function is used to update following Mplus input sections:

- TITLE
- DATA
- DATA IMPUTATION
- DATA WIDETOLONG
- DATA LONGTOWIDE
- DATA TWOPARTE

- DATA MISSING
- DATA SURVIVAL
- DATA COHORT
- VARIABLE
- DEFINE
- ANALYSIS
- MODEL
- MODEL INDIRECT
- MODEL CONSTRAINT
- MODEL TEST
- MODEL PRIORS
- MODEL MONTECARLO
- MODEL POPULATION
- MODEL COVERAGE
- MODEL MISSING
- OUTPUT
- SAVEDATA
- PLOT

The ...; **Specification** The ...; Specification is used to update specific options in the VARIABLE and ANALYSIS section, while keeping all other options in the `misty.object` of type `mplus` specified in the argument `x`. The ...; specification is only available for the VARIABLE and ANALYSIS section. Note that ...; including the semicolon ; needs to be specified, i.e., ... without the semicolon ; will result in an error message.

The −; **Specification** The ---; specification is used to remove entire sections (e.g., OUTPUT: ---;) or options within the VARIABLE: and ANALYSIS: section (e.g., ANALYSIS: ESTIMATOR IS ---;) from the Mplus input. Note that ---; including the semicolon ; needs to be specified, i.e., --- without the semicolon ; will result in an error message.

Comments in the Mplus Input Comments in the Mplus Input can cause problems when following keywords in uppercase, lower case, or mixed upper and lower case letters are involved in the comments of the VARIABLE and ANALYSIS section:

- VARIABLE **section**: "NAMES", "USEOBSERVATIONS", "USEVARIABLES", "MISSING", "CENSORED", "CATEGORICAL", "NOMINAL", "COUNT", "DSURVIVAL", "GROUPING", "IDVARIABLE", "FREQWEIGHT", "TSCORES", "AUXILIARY", "CONSTRAINT", "PATTERN", "STRATIFICATION", "CLUSTER", "WEIGHT", "WTSCALE", "BWEIGHT", "B2WEIGHT", "B3WEIGHT", "BWTSCALE", "REPWEIGHTS", "SUBPOPULATION", "FINITE", "CLASSES", "KNOWNCLASS", "TRAINING", "WITHIN", "BETWEEN", "SURVIVAL", "TIMECENSORED", "LAGGED", or "TINTERVAL".
- ANALYSIS **section**: "TYPE", "ESTIMATOR", "MODEL", "ALIGNMENT", "DISTRIBUTION", "PARAMETERIZATION", "LINK", "ROTATION", "ROWSTANDARDIZATION", "PARALLEL", "REPSE", "BASEHAZARD", "CHOLESKY", "ALGORITHM", "INTEGRATION", "MCSEED", "ADAPTIVE", "INFORMATION", "BOOTSTRAP", "LRTBOOTSTRAP", "STARTS", "STITERATIONS", "STCONVERGENCE", "STSCALE", "STSEED", "OPTSEED", "K-1STARTS", "LRTSTARTS", "RSTARTS", "ASTARTS", "H1STARTS", "DIFFTEST", "MULTIPLIER", "COVERAGE", "ADDFREQUENCY", "ITERATIONS", "SDITERATIONS", "H1ITERATIONS", "MITERATIONS", "MCITERATIONS", "MUITERATIONS", "RITERATIONS", "AITERATIONS", "CONVERGENCE", "H1CONVERGENCE", "LOGCRITERION", "RLOGCRITERION", "MCONVERGENCE", "MCCONVERGENCE", "MUCONVERGENCE", "RCONVERGENCE",

"ACONVERGENCE", "MIXC", "MIXU", "LOGHIGH", "LOGLOW", "UCELLSIZE", "VARIANCE", "SIMPLICITY", "TOLERANCE", "METRIC", "MATRIX", "POINT", "CHAINS", "BSEED", "STVALUES", "PREDICTOR", "ALGORITHM", "BCONVERGENCE", "BITERATIONS", "FBITERATIONS", "THIN", "MDITERATIONS", "KOLMOGOROV", "PRIOR", "INTERACTIVE", or "PROCESSORS".

Note that comments are removed from the input text by default, i.e., comment = FALSE.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
x	misty.object object of type mplus
args	specification of function arguments
input	list with input command sections
write	updated write command sections
result	list with input command sections (input) and result sections (input)

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.mplus, write.mplus, mplus, mplus.print, mplus.plot, mplus.bayes, mplus.run, mplus.lca

Examples

```
## Not run:

#————————————————————————————————————————————————————————————————————
# Example 1: Update VARIABLE and MODEL section

# Write Mplus Data File
write.mplus(ex3_1, file = "ex3_1.dat")

# Specify Mplus input
input <- '
DATA:     FILE IS ex3_1.dat;
VARIABLE: NAMES ARE y1 x1 x3;
MODEL:    y1 ON x1 x3;
OUTPUT:   SAMPSTAT;
'

# Run Mplus input
```

```
mod0 <- mplus(input, file = "ex3_1.inp")

# Update VARIABLE and MODEL section
update1 <- '
VARIABLE: ...;
          USEVARIABLES ARE y1 x1;
MODEL:    y1 ON x1;
'

# Run updated Mplus input
mod1 <- mplus.update(mod0, update1, file = "ex3_1_update1.inp")

#-----------------------------------------------------------------------
# Example 2: Update ANALYSIS section

# Update ANALYSIS section
update2 <- '
ANALYSIS: ESTIMATOR IS MLR;
'

# Run updated Mplus input
mod2 <- mplus.update(mod1, update2, file = "ex3_1_update2.inp")

#-----------------------------------------------------------------------
# Example 3: Remove OUTPUT section

# Remove OUTPUT section
update3 <- '
OUTPUT: ---;
'

# Run updated Mplus input
mod3 <- mplus.update(mod2, update3, file = "ex3_1_update3.inp")

## End(Not run)
```

`multilevel.alpha` *Multilevel Coefficient Alpha*

Description

This function computes point estimate and Monte Carlo confidence interval for the multilevel coefficient alpha defined by Lai (2021) by calling the `cfa` function in the R package **lavaan**. By default, the function prints level-specific multilevel coefficient alpha with 95% Monte Carlo confidence interval based on Huber-White standard errors.

Usage

```
multilevel.alpha(data, ..., cluster,
                 se = c("none", "standard", "robust.huber.white"),
```

```
optim.method = c("nlminb", "em"), missing = c("listwise", "fiml"),
  nrep = 100000, seed = NULL, conf.level = 0.95,
  print = c("all", "alpha", "item"), digits = 2, r.digits = 3,
  as.na = NULL, write = NULL, append = TRUE, check = TRUE,
  output = TRUE)
```

Arguments

`data`	a data frame. A saturated multilevel model is estimated to obtain variances and covariances at the Within and Between level. Note that the cluster variable specified in `cluster` is excluded from `data` when specifying the argument `cluster` using the variable name of the cluster variable.
`...`	an expression indicating the variable names in `data`, e.g., `multilevel.alpha(dat, x1, x2, x3, cluster = "cluster")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`cluster`	either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).
`se`	a character string indicating the standard errors used for computing Monte Carlo confidence intervals, i.e., `"none"` for no standard errors, `"standard"` for conventional standard error based on inverting the expected observed or first.order information matrix, and `"robust.huber.white"` for the 'MLR' (aka pseudo ML, Huber-White) approach. Note that `se = "none"` saves computation time by not computing standard errors and Monte Carlo confidence intervals.
`optim.method`	a character string indicating the optimizer, i.e., `"nlminb"` (default) for the unconstrained and bounds-constrained quasi-Newton method optimizer and `"em"` for the Expectation Maximization (EM) algorithm.
`missing`	a character string indicating how to deal with missing data, i.e., `"listwise"` for listwise deletion or `"fiml"` (default) for full information maximum likelihood (FIML) method.
`nrep`	an integer value indicating the number of Monte Carlo repetitions for computing confidence intervals.
`seed`	a numeric value specifying the seed of the random number generator for computing the Monte Carlo confidence interval.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`print`	a character vector indicating which results to show, i.e. `"all"` (default) for all results `"alpha"` for the composite reliability alpha and `"item"` for item statistics. Note that standardized factor loadings at the within and between level printed when requesting `"item"` are based on a multilevel CFA model with factor loadings freely estimated across both levels.
`digits`	an integer value indicating the number of decimal places to be used for displaying mean, standard deviation, minimum, maximum, skewness, and kurtosis.
`r.digits`	an integer value indicating the number of decimal places to be used for displaying multilevel coefficient alpha, ICC(1), and standardized factor loadings.

`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data` but not to `cluster`.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification, convergence and model identification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Details

In single-level data, coefficient α (Cronbach, 1951) can be computed as

$$\alpha = \frac{2p\Sigma_{k=2}^{p}\Sigma_{k'=1}^{k-1}\sigma_{kk'}}{(p-1)\mathbf{1'\Sigma 1}}$$

assuming that the covariance matrix of p items is Σ with elements $\sigma_{ij'}$. Note that coefficients α and ω represent the same population quantities under unidimensionality and essential tau-equivalence. The multilevel extension of coefficient α, however, does not require estimation of parameters of a factor model. Conseuqntly, its computation at the between-level remains the same regardless a composite is used to measure a configural construct or a shared construct.

Within-Level Composite The reliability for an observed within-level composite, Z_{ij}^{w}, is

$$\alpha^{w} = \frac{2p\Sigma_{k=2}^{p}\Sigma_{k'=1}^{k-1}\sigma_{kk'}^{w}}{(p-1)\mathbf{1'\Sigma}^{w}\mathbf{1}}$$

Between-Level Composite The reliability for an observed between-level composite, Z_{j}^{b}, is

$$\alpha^{b} = \frac{2p\Sigma_{k=2}^{p}\Sigma_{k'=1}^{k-1}\sigma_{kk'}^{b}}{(p-1)[\mathbf{1'\Sigma}^{b}\mathbf{1} + \mathbf{1'\Sigma}^{w}\mathbf{1}/\tilde{n}]}$$

Overall Composite The reliability for an overall observed composite, Z_{ij}, capturing the population variance of both the within-level and the between-level components of the true score and the errors is

$$\alpha^{2l} = \frac{2p\Sigma_{k=2}^{p}\Sigma_{k'=1}^{k-1}(\sigma_{kk'}^{w} + \sigma_{kk'}^{b})}{(p-1)[\mathbf{1'\Sigma}^{b}\mathbf{1} + \mathbf{1'\Sigma}^{w}\mathbf{1}]}$$

Note that α^{2l} is simply the ratio of true variance to total variance of the observed scores. In practice, α^{2l} is not the most interesting coefficient because researchers are mainly interested in distinguishing within- and between effects (Castro-Alvarez et al., 2026).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data` including the group variable specified in `cluster`
`args`	specification of function arguments
`model`	specified model
`model.fit`	fitted lavaan object (`mod.fit`)
`check`	results of the convergence and model identification check
`result`	list with result tables, i.e., `alpha` for the coefficient alpha including Monte Carlo confidence interval and `itemstat` for descriptive statistics

Note

This function is based on the function `multilevel_alpha` from Mark Lai (2021 supplementary codes) and uses the functions `lavInspect`, `lavTech`, `lavNames`, `parameterEstimates`, and `sem` provided in the R package **lavaan** by Yves Rosseel (2012). The internal function `.internal.mvrnorm` is a copy of the `mvrnorm` function in the package **MASS** by Venables and Ripley (2002).

Author(s)

Takuya Yanagida

References

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297-334. http://dx.doi.org/10.1007/BF02310555

Lai, M. H. C. (2021). Composite reliability of multilevel data: It's about observed scores and construct meanings. *Psychological Methods, 26*(1), 90–102. https://doi.org/10.1037/met0000287

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

Venables, W. N., Ripley, B. D. (2002).*Modern Applied Statistics with S* (4th ed.). Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

See Also

`multilevel.omega`, `item.omega`, `multilevel.cfa`, `multilevel.fit`, `multilevel.invar`, `multilevel.cor`, `multilevel.descript`, `write.result`

Examples

```
## Not run:
# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————
```

```
# Cluster Variable Specification

# Example 1a: Specification using the argument '...'
multilevel.alpha(Demo.twolevel, y1:y4, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.alpha(Demo.twolevel[, c("y1", "y2", "y3", "y4", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.alpha(Demo.twolevel[, c("y1", "y2", "y3", "y4")], cluster = Demo.twolevel$cluster)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Arguments 'se' and 'print'

# Example 4a: No confidence intervals to speed up computation
multilevel.alpha(Demo.twolevel, y1:y4, cluster = "cluster", se = "none")

# Example 4b: Alpha and item statistics
multilevel.alpha(Demo.twolevel, y1:y4, cluster = "cluster", print = "all")

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Write Results

# Example 5a: Write results into a text file
multilevel.alpha(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster",
                 write = "Multilevel_Alpha.txt")

# Example 5b: Write results into a Excel file
multilevel.alpha(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster",
                 write = "Multilevel_Alpha.xlsx")

## End(Not run)
```

`multilevel.cfa` *Multilevel Confirmatory Factor Analysis*

Description

This function conducts multilevel confirmatory factor analysis to investigate different types of constructs with distinct construct meanings at the individual and cluster level (Stapleton et al., 2016) by calling the `cfa` function in the R package **lavaan**. By default, the function specifies and estimates a measurement model for an individual and configural construct to provides a table with univariate sample statistics, model fit information, and parameter estimates. Additionally, variance-covariance coverage of the data, modification indices, residual correlation matrix, and relative Opdyke distribution percentile matrix can be requested by specifying the argument `print`.

Usage

```
multilevel.cfa(data, ..., cluster, model = NULL, rescov = NULL,
               model.w = NULL, model.b = NULL, rescov.w = NULL, rescov.b = NULL,
```

```
    const = c("within", "shared", "config", "shareconf"),
    fix.resid = NULL, ident = c("marker", "var", "effect"),
    ls.fit = FALSE, estimator = c("ML", "MLR"),
  test = c("none", "standard", "yuan.bentler", "yuan.bentler.mplus"),
    se = c("none", "standard", "robust.huber.white"),
   optim.method = c("nlminb", "em"), missing = c("listwise", "fiml"),
   print = c("all", "summary", "coverage", "descript", "fit", "est",
              "modind", "resid", "opdyke"),
    mod.minval = 6.63, resid.minval = 0.1, opdyke.prec = 1,
    opdyke.minmax = c(0.40, 0.60), color = "b.red",
   style = c("regular", "bold", "italic"), digits = 3, p.digits = 3,
as.na = NULL, write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. If `model`, `model.w`, and `model.b` are `NULL`, multilevel confirmatory factor analysis based on a measurement model with one factor labeled `wf` at the Within level and one factor labeled `bf` at the Between level comprising all variables in the data frame is conducted. Note that the cluster variable specified in `cluster` is excluded from `data` when specifying the argument `cluster` using the variable name of the cluster variable. If `model` or `mode.w` and `model.b` is specified, the data frame needs to contain all variables used in the `model` argument(s).

`...` an expression indicating the variable names in `data`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`cluster` either a character string indicating the variable name of the cluster variable in `data` or `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).

`model` a character vector for specifying the same factor structure with one factor at the Within and Between Level, or a list of character vectors for specifying the same measurement model with more than one factor at the Within and Between Level, e.g.,`model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `wf` at the Within level and a measurement model with one factor labeled `bf` at the Between level each comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"),factor2 = c("x5", "x6", "x7", "x8"))` for specifying a measurement model with two latent factors labeled `wfactor1` and `wfactor2` at the Within level and a measurement model with two latent factors labeled `bfactor1` and `bfactor2` at the Between level each comprising four indicators. Note that the name of each list element is used to label factors, where prefixes `w` and `b` are added the labels to distinguish factor labels at the Within and Between level, i.e., all list elements need to be named, otherwise factors are labeled with `"wf1"`, `"wf2"`, `"wf3"` for labels at the Within level and `"bf1"`, `"bf2"`, `"bf3"` for labels at the Between level and so on.

`rescov` a character vector or a list of character vectors for specifying residual covariances at the Within level, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between indicators `x1` and `x2` at the Within level or `rescov =`

list(c("x1", "x2"), c("x3", "x4")) for specifying residual covariances between indicators x1 and x2, and indicators x3 and x4 at the Within level. Note that residual covariances at the Between level can only be specified by using the arguments model.w, model.b, and model.b.

model.w — a character vector specifying a measurement model with one factor at the Within level, or a list of character vectors for specifying a measurement model with more than one factor at the Within level.

model.b — a character vector specifying a measurement model with one factor at the Between level, or a list of character vectors for specifying a measurement model with more than one factor at the Between level.

rescov.w — a character vector or a list of character vectors for specifying residual covariances at the Within level.

rescov.b — a character vector or a list of character vectors for specifying residual covariances at the Between level.

const — a character string indicating the type of construct(s), i.e., "within" for within-cluster constructs, "shared" for shared cluster-level constructs, "config" (default) for configural cluster constructs, and "shareconf" for simultaneous shared and configural cluster constructs.

fix.resid — a character vector for specifying residual variances to be fixed at 0 at the Between level, e.g., fix.resid = c("x1", "x3") to fix residual variances of indicators x1 and x2 at the Between level at 0. Note that it is also possible to specify fix.resid = "all" which fixes all residual variances at the Between level at 0 in line with the strong factorial measurement invariance assumption across cluster.

ident — a character string indicating the method used for identifying and scaling latent variables, i.e., "marker" for the marker variable method fixing the first factor loading of each latent variable to 1, "var" for the fixed variance method fixing the variance of each latent variable to 1, or "effect" for the effects-coding method using equality constraints so that the average of the factor loading for each latent variable equals 1.

ls.fit — logical: if TRUE (default) level-specific fit indices are computed when specifying a model using the arguments model.w and model.b given the model does not contain any cross-level equality constraints.

estimator — a character string indicating the estimator to be used: "ML" for maximum likelihood with conventional standard errors and "MLR" (default) for maximum likelihood with Huber-White robust standard errors and a scaled test statistic that is asymptotically equal to the Yuan-Bentler test statistic. Note that by default, full information maximum likelihood (FIML) method is used to deal with missing data when using "ML" (missing = "fiml"), whereas incomplete cases are removed listwise (i.e., missing = "listwise") when using "MLR".

test — a character string indicating the test statistics, i.e., "none" for no test statistic, "standard" for a conventional chi-square test, "yuan.bentler" for a Yuan-Bentler scaled test statistic, and "yuan.bentler.mplus" for a test statistic which is asymptotically equivalent to the Yuan-Bentler T2-star test statistic. Note that the default setting is depending on the argument estimator, i.e., test =

"standard" when estimator = "ML" and test = "yuan.bentler.mplus" when estimator = "MLR".

se — a character string indicating the standard errors, i.e., "none" for no standard errors, "standard" for conventional standard error based on inverting the expected observed or first.order information matrix, and "robust.huber.white" for the 'MLR' (aka pseudo ML, Huber-White) approach. Note that the default setting is depending on the argument estimator, i.e., test = "standard" when estimator = "ML" and test = "robust.huber.white" when estimator = "MLR".

optim.method — a character string indicating the optimizer, i.e., "nlminb" (default) for the unconstrained and bounds-constrained quasi-Newton method optimizer and "em" for the Expectation Maximization (EM) algorithm.

missing — a character string indicating how to deal with missing data, i.e., "listwise" (default) for listwise deletion or "fiml" for full information maximum likelihood (FIML) method. Note that FIML method is only available when estimator = "ML", that it takes longer to estimate the model using FIML, and that FIML is prone to convergence issues which might be resolved by switching to listwise deletion.

print — a character vector indicating which results to show on the console, i.e. "all" for all results, "summary" for a summary of the specification of the estimation method and missing data handling in lavaan, "coverage" for the variance-covariance coverage of the data, "descript" for descriptive statistics, "fit" for model fit, "est" for parameter estimates, "modind" for modification indices, "resid" for the residual correlation matrix and standardized residual means, and "opdyke" for the discrepancy between model-implied and observed correlation expressed in terms of the relative percentile of an Opdyke distribution (McNeish, 2025). By default, a summary of the specification, model fit, and parameter estimates are printed. By default, a summary of the specification, descriptive statistics, model fit, and parameter estimates are printed.

mod.minval — numeric value to filter modification indices and only show modifications with a modification index value equal or higher than this minimum value. By default, modification indices equal or higher 6.63 are printed. Note that a modification index value of 6.63 is equivalent to a significance level of $\alpha = .01$.

resid.minval — numeric value indicating the minimum absolute residual correlation coefficients and standardized means to highlight. By default, absolute residual correlation coefficients and standardized means equal or higher 0.1 are highlighted. Note that highlighting can be disabled by setting the minimum value to 1 or by setting the argument color to "default".

opdyke.prec — a numeric value indicating the precision of the probability density function calculations of the Opdyke distribution. The default is 1 which calculates the PDF for polar angles between $(0, pi)$ in 0.01 increments. Specifying 10 calculates the PDF polar angles between $(0, pi)$ in 0.001 increments, which takes considerably longer, especially if there are many correlation elements.

opdyke.minmax — a numeric vector with two elements indicating the minimum and maximum percentile of the Opdyke distribution that is considered to be acceptably close to

the observed correlation represented by the Opdyke distribution median. Predicted correlation outside the range will be color highlighted in line with to the argument `color`.

`color` a character string indicating the text color for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval` and predicted correlations outside the minimum and maximum percentile of the Opdyke distribution, i.e., i.e., `"default"` for the default text color without color coding and various text colors for highlighting `"red"`, `"b.red"` (default), `"green"`, `"b.green"`, `"blue"`, or `"b.blue"`, see the help page of the `chr.color` function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to `"default"` when using R Markdown and when writing the output into a text file (`.txt`).

`style` a character vector indicating the font style for highlighting absolute residual correlation coefficients and standardized means equal or higher `resid.minval`, i.e., `"regular"` (default) for regular text, `"bold"` for bold text, and `"italic"` for italic text. Note that the font style `"bold"` and `"italic"` can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument `color` needs to be specified to change the style of the text, e.g. `color = "black"` and `style = "bold"` to for bold text.

`digits` an integer value indicating the number of decimal places to be used for displaying results. Note that loglikelihood, information criteria and chi-square test statistic is printed with `digits` minus 1 decimal places.

`p.digits` an integer value indicating the number of decimal places to be used for displaying the *p*-value.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data` but not to `cluster`.

`write` a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.

`append` logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.

`check` logical: if `TRUE` (default), argument specification, convergence and model identification is checked.

`output` logical: if `TRUE` (default), output is shown.

Details

Stapleton et al. (2016) identified different types of constructs when data are collected from individuals nested within clusters characterized by different interpretations of construct meaning. At the individual level, the hypothesized construct is expected to be relevant to individual item responses, i.e., scores on the measure reflect individual variability. At the cluster level, there are two types of constructs measured using item responses from individuals: *shared construct*, a characteristic of the cluster itself and *configural construct*, which reflects a construct at the individual level. Marsh et al. (2012) introduced the terms *climate construct* and *contextual construct*, while Lüdtke et al.,

(2011) used the term *reflective construct* and *formative construct* to differentiate between shared and configural constructs.

Five types of confirmatory factor analysis (CFA) models were proposed for construct validation with nested data, depending on whether the construct of interest exists solely at the individual level or is also relevant at the cluster level (Stapleton et al., 2016).

Model 1: Individual Constructs, Single-Level Model Model 1 is a single-level CFA model whose parameter estimates provide an aggregate of the relationship between item responses and the individual construct. A design-based estimation approach is used to adjust the standard error estimates, chi-square test statistic, and fit indices to account for the dependency of item responses among individuals within clusters. Note that this model is estimated using the `item.cfa` with specifying the `cluster` argument.

This model is appropriate when a positive ICC(1) is spurious, resulting from a selection process. For example, lactose intolerance is an entirely individual characteristic, unaffected by environment. However, item responses measuring lactose intolerance might exhibit a non-negligible ICC(1) due to the selection of individuals across different clusters, rather than reflecting a true cluster-level construct. That is, lactose intolerance is a completely individual measure, not able to be influenced by environmental effect.

The model is designed for application across a broad population that may or may not be nested within clusters in a single-level model. However, if the measure aims to compare the relative positions of individuals within a cluster, a within-cluster approach is more suitable.

Model 2: Within-Cluster Constructs Model 2 is a two-level CFA model specified by `const = "within"` that includes a within-cluster construct representing an individual's standing within the cluster. The model does not assume the existence of a cluster-level construct but allows for cluster-level variability in each measure through a saturated variance-covariance model.

The model is designed for application within clusters to compare individuals and their relative positions within a specific cluster, while between-cluster constructs are not meaningful or not of interest.

Model 3: Shared Cluster-Level Constructs Model 3 is a two-level CFA model specified by `const = "shared"` that includes a between-cluster construct representing a cluster-level characteristic that is measured using individuals within a cluster as the information source. The model does not assume the existence of an individual-level construct but allows for individual-level variability in each measure through a saturated variance-covariance model.

For a shared construct, individual item responses should be interchangeable, reflecting perfect interrater reliability and complete within-cluster agreement. However, within-cluster variation can exist for shared constructs, as these constructs stem from the experiences, attitudes, perceptions, values, or cognitions of individual cluster members. Given evidence of adequate agreement within clusters, the aggregate value of the measure can be assigned to the cluster, i.e., the shared perceptions of individuals in a cluster are interpreted as a proxy for the cluster-level characteristic (Jak et al., 2023). From this perspective, the greater the agreement within the cluster, the more reliably the shared construct is measured, although perfect agreement is not required. A reasonable step in the shared construct modeling process is to examine the ICC(2) values for each item measuring the shared construct. Klein et al. (2000) suggested that ICC(2) values of at least 0.7 indicate acceptable reliability of the measured shared construct, values between 0.5 and 0.7 indicate marginal reliability, and values below 0.5 are considered poor.

Note that the wording of questionnaire items plays a crucial role in the context of multilevel measurement. For example, the statement "The instructor presents materials in ways that keep

it interesting." would likely to elicit responses reflecting the instructor's qualities, whereas the statement "I find the class meeting interest." would likely to reflect both a characteristic of the cluster and the individual's intrinsic interest in the class topic. Accordingly, an argument should be made regarding how the item wording targets cluster characteristics.

Model 4: Individual and Configural Constructs Model 4 is a two-level CFA model specified by `const = "config"` that includes a within-cluster construct representing an individual's standing within the cluster and a between-cluster construct representing an aggregate of the measurements of individuals within the cluster. The configural cluster construct model requires cross-level measurement invariance of factor loadings to interpret the individual- and cluster-level common factors as reflecting the individual- and cluster-level components of the same construct (Jak et al., 2023).

For a configural construct, individual item responses are not interchangeable, and it is not expected that individuals within a cluster respond similarly to the items. Marsh et al. (2012) noted that if item responses have an ICC(1) of zero, there is little justification for continuing to examine the configural cluster-level construct. However, Stapleton et al. (2016) argued that it may still be of interest to investigate the variability in individual hypothetical latent scores across clusters, i.e., the dispersion of the construct may differ across clusters and represent an important cluster characteristic.

Model 5: Simultaneous Shared and Configural Constructs Model 5 is a two-level CFA model specified by `const = "shareconf"` that includes a within-cluster construct representing an individual's standing within the cluster and two between-cluster constructs: a configural construct representing the individual-level construct, and a shared construct modeling additional covariation among cluster-average item responses, representing rater effects. The factor loadings of the configural construct are constrained to be equal across levels, i.e., metric cross-level measurement invariance.

For example, suppose that teachers have provided ratings of their students' motivation using multiple items for each child. It is possible that some teachers rate more positively compared to others, resulting in two sources of covariation at the cluster level: variation due to the fact that students in some classes are truly more motivated on average than those in other classes, and variation due to rater effects, because some teachers tend to rate their students more positively or more negatively on average.

Note that what may appear to be an additional shared construct could instead reflect a spurious contextual effect. Specifically, when constraining factor loadings across levels results in a relatively poor fit compared to a model with unconstrained loadings in a configural model, this suggests that the apparent additional shared construct is actually due to measurement noninvariance at the within-cluster level.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`model`	specified model
`model.fit`	fitted lavaan object (`mod.fit`)

`check` results of the convergence and model identification check

`result` list with result tables, i.e., `summary` for the summary of the specification of the estimation method and missing data handling in lavaan, `coverage` for the variance-covariance coverage of the data, `descript` for descriptive statistics, `fit` for model fit, `est` for a list with parameter estimates for Within and Between, `score` fir a list with modification indices for parameter constraints for Within and Between, `resid` for a list with residual correlation matrices and standardized residual means for the Within and Between level, and `opdyke` for the relative Opdyke distribution percentile matrix for the Within and Between level.

Note

The function uses the functions `cfa`, `lavInspect`, `lavTech`, `modindices`, `parameterEstimates`, and `standardizedsolution` provided in the R package **lavaan** by Yves Rosseel (2012). The relative Opdyke distribution percentile matrix is based on the functions `opdyke` and `opdyke.percentiles` provided in the R package **opddyke** by Dan McNeish (2026).

Author(s)

Takuya Yanagida

References

Jak, S., Jorgensen, T. D., ten Hove, D., & Nevicka, B. (2023). Modeling cluster-level constructs measured by individual responses: Configuring a shared approach. *Advances in Methods and Practices in Psychological Science, 6*(3), Article 25152459231182319. https://doi.org/10.1177/25152459231182319

Klein, K. J., & Kozlowski, S. W. (2000). Multilevel theory, research, and methods in organizations: Foundations, extensions, and new directions. Jossey-Bass.

Lüdtke, O., Marsh, H. W., Robitzsch, A., & Trautwein, U. (2011). A 2x2 taxonomy of multilevel latent contextual models: Accuracy-bias trade-offs in full and partial error correction models. *Psychological Methods, 16*, 444-467. https://doi.org/10.1037/a0024376

Marsh, H. W., Ludtke, O., Nagengast, B., Trautwein, U., Morin, A. J. S., Abduljabbar, A. S., & Koller, O. (2012). Classroom climate and contextual effects: Conceptual and methodological issues in the evaluation of group-level effects. *Educational Psychologist, 47*, 106-124. https://doi.org/10.1080/00461520.2012.670488

McNeish, D. (2025). Less Heuristic Approximate Local Fit Evaluation in Structural Equation Models. *Structural Equation Modeling: A Multidisciplinary Journal, 32*(4), 590-605. https://doi.org/10.1080/10705511.2025.2473342

McNeish D (2026). opdyke: Opdyke Percentiles for Approximate Local Fit in Structural Equation Models. R package version 1.0.0. Retrieved from https://github.com/dmcneish18/opdyke

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

Stapleton, L. M., Yang, J. S., & Hancock, G. R. (2016). Construct meaning in multilevel settings. *Journal of Educational and Behavioral Statistics, 41*, 481-520. http://dx.doi.org/10.3102/1076998616646200

See Also

item.cfa, multilevel.fit, multilevel.invar, multilevel.omega, multilevel.cor, multilevel.descript

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Model Specification Using 'data' for a One-Factor Model
#
# Same factor structure with one factor at the Within and Between Level

#————————————————————————————————————
## Cluster Variable Specification

# Example 1a: Specification using the argument '...'
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.cfa(Demo.twolevel[, c("y1", "y2", "y3", "y4", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.cfa(Demo.twolevel[, c("y1", "y2", "y3", "y4")], cluster = Demo.twolevel$cluster)

#————————————————————————————————————
## Type of Construct

# Example 2a: Within-cluster constructs
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", const = "within")

# Example 2b: Shared cluster-level construct
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", const = "shared")

# Example 2c: Individual and configural cluster construct (default)
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", const = "config")

# Example 2d: Simultaneous shared and configural cluster construct
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", const = "shareconf")

#————————————————————————————————————
## Residual Covariances at the Within Level

# Example 3a: Residual covariance between 'y1' and 'y3'
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", rescov = c("y1", "y3"))

# Example 3b: Residual covariance between 'y1' and 'y3', and 'y2' and 'y4'
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster",
               rescov = list(c("y1", "y3"), c("y2", "y4")))
```

```
#————————————————————————————————————
## Residual Variances at the Between Level fixed at 0

# Example 4a: All residual variances fixed at 0
# i.e., strong factorial invariance across clusters
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", fix.resid = "all")

# Example 4b: Residual variances of 'y1', 'y2', and 'y4' fixed at 0
# i.e., partial strong factorial invariance across clusters
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", fix.resid = c("y1", "y2"))

#————————————————————————————————————
## Arguments 'print', 'mod.minval', 'resid.minval', and 'opdyke.minmax'

# Example 5a: Request all results
item.cfa(Demo.twolevel, y1:y6, cluster = "cluster", print = "all")

# Example 5b: Request modification indices with value equal or higher than 2
item.cfa(Demo.twolevel, y1:y6, cluster = "cluster", print = "modind", mod.minval = 2)

# Example 5c: Highlight absolute residual correlation equal or higher than 0.05
item.cfa(Demo.twolevel, y1:y6, cluster = "cluster", print = "resid", resid.minval = 0.05,
         color = "b.blue")

# Example 5d: Highlight Opdyke distribution percentiles outside 0.45 and 0.55
item.cfa(Demo.twolevel, y1:y6, cluster = "cluster", print = "opdyke", opdyke.minmax = c(0.45, 0.55),
         color = "black", style = "bold")

#————————————————————————————————————————————————————————————————————
# Model Specification using 'model' for One or Multiple-Factor Model
#
# Same factor structure at the Within and Between Level

# Example 6a: One-factor model
multilevel.cfa(Demo.twolevel, cluster = "cluster", model = c("y1", "y2", "y3", "y4"))

# Example 6b: Two-factor model
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))

# Example 6c: Two-factor model with user-specified labels for the factors
multilevel.cfa(Demo.twolevel, cluster = "cluster",
              model = list(factor1 = c("y1", "y2", "y3"), factor2 = c("y4", "y5", "y6")))

#————————————————————————————————————
# Type of Construct

# Example 7a: Within-cluster constructs
multilevel.cfa(Demo.twolevel, cluster = "cluster", const = "within",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))

# Example 7b: Shared cluster-level construct
multilevel.cfa(Demo.twolevel, cluster = "cluster", const = "shared",
```

```
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))

# Example 7c: Individual and configural cluster construct (default)
multilevel.cfa(Demo.twolevel, cluster = "cluster", const = "config",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))

# Example 7d: Simultaneous shared and configural cluster construct
multilevel.cfa(Demo.twolevel, cluster = "cluster", const = "shareconf",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))

#————————————————————————————————————
## Residual Covariances at the Within Level

# Example 8a: Residual covariance between 'y1' and 'y4' at the Within level
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")),
               rescov = c("y1", "y4"))

# Example 8b: Fix all residual variances at 0
# i.e., strong factorial invariance across clusters
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")),
               fix.resid = "all")

#————————————————————————————————————————————————————————————————————————
# Model Specification Using 'model.w' and 'model.b' for One or Multiple_Factor Model
#
# Different factor structure at the Within and Between Level

# Example 9a: Simultaneous and level-specific fit indices
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model.w = c("y1", "y2", "y3", "y4", "y5", "y6"),
               model.b = c("y1", "y2", "y3", "y4", "y5", "y6"), ls.fit = TRUE)

# Example 9b: Two-factor model at the Within level and one-factor model at the Between level
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model.w = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")),
               model.b = c("y1", "y2", "y3", "y4", "y5", "y6"))

# Example 9c: Residual covariance between 'y1' and 'y4' at the Within level
# Residual covariance between 'y5' and 'y6' at the Between level
multilevel.cfa(Demo.twolevel, cluster = "cluster",
               model.w = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")),
               model.b = c("y1", "y2", "y3", "y4", "y5", "y6"),
               rescov.w = c("y1", "y4"),
               rescov.b = c("y5", "y6"))

#————————————————————————————————————————————————————————————————————————
# lavaan Model and Summary of the Estimated Model

# Example 10
mod <- multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", output = FALSE)
```

```
# lavaan model syntax
cat(mod$model)

# Fitted lavaan object
lavaan::summary(mod$model.fit, standardized = TRUE, fit.measures = TRUE)

#-----------------------------------------------------------------------
# Write Results

# Example 11a: Write results into a text file
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", print = "all",
                      write = "Multilevel_CFA.txt", output = FALSE)

# Example 11b: Write results into an Excel file
multilevel.cfa(Demo.twolevel, y1:y4, cluster = "cluster", print = "all",
               write = "Multilevel_CFA.xlsx", output = FALSE)

## End(Not run)
```

`multilevel.cor` *Within-Group and Between-Group Correlation Matrix*

Description

This function computes the within-group and between-group correlation matrix by calling the `sem` function in the R package **lavaan** and provides standard errors, z test statistics, and significance values (p-values) for testing the hypothesis H0: $\rho = 0$ for all pairs of variables within and between groups. By default, the function provides the within-group and between-group correlation matrix. Statistically significant correlations can be highlighted by specifying the argument `color`.

Usage

```
multilevel.cor(data, ..., cluster, estimator = c("ML", "MLR"), constr.var = FALSE,
                optim.method = c("nlminb", "em"), optim.switch = TRUE,
             print = c("all", "summary", "cor", "se", "stat", "p"), split = FALSE,
              order = FALSE, tri = c("both", "lower", "upper"), tri.lower = TRUE,
                missing = c("listwise", "fiml"), alpha = 0.05,
                color = "default", style = c("regular", "bold", "italic"),
                p.adj = "none", digits = 2, p.digits = 3, as.na = NULL,
                plot = FALSE, geom = c("none", "label"), filename = NULL,
                width = NA, height = NA, dpi = 600, write = NULL, append = TRUE,
                check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame.
`...`	an expression indicating the variable names in `data`, e.g., `multilevel.cor(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`cluster` either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).

`estimator` a character string indicating the estimator to be used, i.e., `"ML"` for maximum likelihood with conventional standard errors and `"MLR"` for maximum likelihood with Huber-White robust standard errors. The default setting depends on the argument `color`, i.e., `"ML"` is used when specifying `color = "default"` (default) and `"MLR"` is used when specifying a color for the argument `color`.

`constr.var` logical: if `TRUE`, inequality constraints are imposed for the variance parameters at the between level, i.e., variances are constrained to be greater than 0.

`optim.method` a character string indicating the optimizer, i.e., `"nlminb"` for the unconstrained and bounds-constrained quasi-Newton method optimizer and `"em"` for the Expectation Maximization (EM) algorithm. By default, the optimizer `"nlminb"` is used with complete data or when using listwise deletion for missing data handling, while the optimizer `"em"` is used when using for full information maximum likelihood (FIML) method for missing data handling.

`optim.switch` logical: if `TRUE` (default), model estimation switches to Expectation Maximization (EM) algorithm (`"em"`) if the quasi-Newton optimization (`"nlminb"`) does not converge.

`print` a character vector indicating which results to show on the console, i.e. `"all"` for all results, `"summary"` for a summary of the specification of the estimation method and missing data handling in lavaan, `"cor"` (default) for correlation coefficients, `"se"` for standard errors, `"stat"` for z test statistics, and `"p"` for *p*-values. By default, the function only prints the within-group and between-group correlation matrix.

`split` logical: if `TRUE`, output table is split in within-group and between-group correlation matrix.

`order` logical: if `TRUE`, variables in the output table are ordered, so that variables specified in the argument `between` are shown first.

`tri` a character string indicating which triangular of the matrix to show on the console when `split = TRUE`, i.e., `both` for upper and `upper` for the upper triangular.

`tri.lower` logical: if `TRUE` (default) and `split = FALSE` (default), within-group correlations are shown in the lower triangular and between-group correlation are shown in the upper triangular.

`missing` a character string indicating how to deal with missing data, i.e., `"listwise"` for listwise deletion or `"fiml"` (default) for full information maximum likelihood (FIML) method. Note that it takes longer to estimate models while using FIML and using FIML is prone to issues with model convergence, these issues might be resolved by switching to listwise deletion.

`alpha` a numeric value between 0 and 1 indicating the significance level at which correlation coefficients are printed boldface when when specifying the argument `color`.

`color` a character string indicating the text color for highlighting statistically significant correlation coefficients , i.e., `"default"` (default) for the default text color without color coding and various text colors for highlighting like `"red"`, `"b.red"`,

"green", "b.green", "blue", or "b.blue", see the help page of the chr.color function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to "default" when using R Markdown and when writing the output into a text file (.txt).

style — a character vector indicating the font style for statistically significant correlation coefficients, i.e., "regular" (default) for regular text, "bold" for bold text, and "italic" for italic text. Note that the font style "bold" and "italic" can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument color needs to be specified to change the style of the text, e.g. color = "black" and style = "bold" to for bold text.

p.adj — a character string indicating an adjustment method for multiple testing based on p.adjust, i.e., none (default), bonferroni, holm, hochberg, hommel, BH, BY, or fdr. See help page of the p.adjust function.

digits — an integer value indicating the number of decimal places to be used for displaying correlation coefficients.

p.digits — an integer value indicating the number of decimal places to be used for displaying *p*-values.

as.na — a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data but not to cluster.

plot — logical: if TRUE,a correlation heat map is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.

geom — character vector indicating which plot elements to draw, i.e., "none" for not printing correlation coefficients inside every tile, "label" for printing correlation coefficients inside every tile.

filename — a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.

width — a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.

height — a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.

dpi — a numeric value indicating the dpi argument (default is 600) for the ggsave function.

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification is checked.

output — logical: if TRUE (default), output is shown on the console.

Details

Within-Group and Between-Group Variables The function automatically identifies (1) variables in the data frame specified in `data` that are measured at the individual level and modeled only at the within level and (2) variables in the data frame specified in `data` that are measured at the cluster level and modeled only at the between level. The former variables have no variance in the between part of the model (i.e., ICC(1) is 0 e.g. due to centering within clusters), while the latter variables do not have any variance within clusters.

Estimation Method The default setting for the argument `estimator` is depending on the setting of the argument `color`. If `color = "default"` (default), maximum likelihood estimation (`estimator = "ML"`) is used, while maximum likelihood with Huber-White robust standard errors (`estimator = "MLR"`) that are robust against non-normality is used when when specifying a color for the argument `color`.

Missing Data Handling In the presence of missing data, full information maximum likelihood (FIML) method (`missing = "fiml"`) is used by default. Note that FIML method cannot deal with within-group variables that have no variance within some clusters. In this cases, the function will switch to listwise deletion. Using FIML method might result in issues with model convergence, which might be resolved by switching to listwise deletion (`missing = "listwise"`).

Optimizer The lavaan package uses a quasi-Newton optimization method (`"nlminb"`) by default. If the optimizer does not converge, model estimation switches to the Expectation Maximization (EM) algorithm (`"em"`) if the argument `optim.switch` is specified as `TRUE` (default).

Statistical Significance Statistically significant correlation coefficients can be shown color coded on the console by specifying the argument `color`. However, this option is not supported when using R Markdown.

Adjustment Method for Multiple Testing Adjustment method for multiple testing when specifying the argument `p.adj` is applied to the within-group and between-group correlation matrix separately.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data` including the group variable specified in `cluster`
`args`	specification of function arguments
`model.fit`	fitted lavaan object (`mod.fit`)
`result`	list with result tables, i.e., `summary` for the specification of the estimation method and missing data handling in lavaan, `wb.cor` for the within- and between-group correlations, `wb.se` for the standard error of the within- and between-group correlations, `wb.stat` for the test statistic of within- and between-group correlations, `wb.p` for the significance value of the within- and between-group correlations, `with.cor` for the within-group correlations, `with.se` for the standard error of the within-group correlations, `with.stat` for the test statistic of within-group correlations, `with.p` for the significance value of the within-group correlations, `betw.cor` for the between-group correlations, `betw.se` for the standard error of the between-group correlations, `betw.stat` for the test statistic of

between-group correlations, `betw.p` for the significance value of the between-group correlations

Note

The function uses the functions `sem`, `lavInspect`, `lavMatrixRepresentation`, `lavTech`, `parameterEstimates`, and `standardizedsolution` provided in the R package **lavaan** by Yves Rosseel (2012).

Author(s)

Takuya Yanagida

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

`write.result`, `multilevel.descript`, `multilevel.icc`, `cluster.scores`

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————
# Cluster Variable Specification

# Example 1a: Specification using the argument '...'
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")], cluster = Demo.twolevel$cluster)

#————————————————————————————————————————————————————————————————————
# Arguments 'color', 'style', 'split', 'print', and 'p.adj'

# Example 2a: Highlight statistically significant result in bright red
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", color = "b.red")

# Example 2b: Highlight statistically significant result in boldface
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", color = "black",
               style = "bold")
```

```
# Example 3: Split output table in within-group and between-group correlation matrix
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", split = TRUE,
               color = "green", style = "bold")

# Example 4a: Print summary of the lavaan specification and all results
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", print = "all")

# Example 4b: Print summary of the lavaan specification and correlation coefficients
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", print = c("summary", "cor"))

# Example 5: Significance values with Bonferroni correction
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", print = c("cor", "p"),
               p.adj = "bonferroni")

#————————————————————————————————————————————————————————————————————————
# Variables Measured at the Within and Cluster Level

# Example 6a: Variables "y1", "y2", and "y2" modeled at both the within and between level
#             Variables "w1" and "w2" modeled at the cluster level
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster")

# Example 6b: Print cluster level variables first
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster", order = TRUE)

#————————————————————————————————————————————————————————————————————————
# lavaan Model and Summary of the Estimated Model

# Example 7: lavaan model and summary of the multilevel model used to compute
#             the within-group and between-group correlation matrix
mod <- multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", output = FALSE)

# lavaan model syntax
mod$model

# Fitted lavaan object
lavaan::summary(mod$model.fit, standardized = TRUE)

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 8a: Correlation heat map
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster", plot = TRUE)

# Example 8b: Split correlation heat map in within-p and between-group
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster",
               split = TRUE, plot = TRUE)

# Example 8c: Highlight statistically significant result in blue
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster",
               color = "blue", plot = TRUE)

# Example 8d: Highlight statistically significant result in boldface
```

```
multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster",
                color = "black", style = "bold", plot = TRUE)

# Example 8e: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <-  multilevel.cor(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster")
plot(object, scale.fill.low = "#DC267F", scale.fill.high = "#009E73",
     text.col = c("gray80", "red"), text.size = 4.5, axis.text.size = 13)

#———————————————————————————————————————————————————————————————————————
# Write Results

# Example 9a: Write Results into a text file
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster",
               write = "Multilevel_Correlation.txt")

# Example 9b: Write Results into a Excel file
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster",
               write = "Multilevel_Correlation.xlsx")

# Example 9c: Save plot as PNG fine
multilevel.cor(Demo.twolevel, y1, y2, y3, cluster = "cluster", plot = TRUE,
               filename = "Multilevel_Correlation.png", width = 8, height = 7)

## End(Not run)
```

`multilevel.descript` *Multilevel Descriptive Statistics for Two-Level and Three-Level Data*

Description

This function computes descriptive statistics for two-level and three-level multilevel data, e.g. average cluster size, variance components, intraclass correlation coefficient, design effect, and effective sample size.

Usage

```
multilevel.descript(data, ..., cluster, type = c("1a", "1b"),
                    method = c("lme4", "nlme"),
                    print = c("all", "var", "sd"), REML = TRUE, digits = 2,
                    icc.digits = 3, as.na = NULL, write = NULL, append = TRUE,
                    check = TRUE, output = TRUE)
```

Arguments

`data`	a numeric vector or data frame.
`...`	an expression indicating the variable names in `data`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`cluster` a character string indicating the name of the cluster variable in `data` for two-level data, a character vector indicating the names of the cluster variables in `data` for three-level data, or a vector or data frame representing the nested grouping structure (i.e., group or cluster variables). Alternatively, a character vector indicating the variable name(s) of the cluster variable(s) in `data`. Note that the cluster variable at Level 3 come first in a three-level model, i.e., `cluster = c("level3", "level2")`.

`type` a character string indicating the type of intraclass correlation coefficient, i.e., `type = "1a"` (default) for ICC(1) representing the proportion of variance at Level 2 and Level 3, `type = "1b"` representing an estimate of the expected correlation between two randomly chosen elements in the same group when specifying a three-level model (i.e., two cluster variables). See 'Details' in the `multilevel.icc` function for the formula used in this function.

`method` a character string indicating the method used to estimate intraclass correlation coefficients, i.e., `"lme4"` (default) ICC estimated using the `lmer` function in the **lme4** package, `"nlme"` ICC estimated using the `lme` function in the **nlme** package.

`print` a character vector indicating which results to show on the console, i.e. `"all"` for variances and standard deviations, `"var"` (default) for variances, or `"sd"` for standard deviations within and between clusters.

`REML` logical: if `TRUE` (default), restricted maximum likelihood is used to estimate the null model when using the `lmer()` function in the **lme4** package or the `lme()` function in the **nlme** package.

`digits` an integer value indicating the number of decimal places to be used.

`icc.digits` an integer indicating the number of decimal places to be used for displaying intraclass correlation coefficients.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data` but not to `cluster`.

`write` a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.

`append` logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.

`check` logical: if `TRUE` (default), argument specification is checked.

`output` logical: if `TRUE` (default), output is shown on the console.

Details

Two-Level Model In a two-level model, the intraclass correlation coefficients, design effect, and the effective sample size are computed based on the random intercept-only model:

$$Y_{ij} = \gamma_{00} + u_{0j} + r_{ij}$$

where the variance in Y is decomposed into two independent components: $\sigma^2_{u_0}$, which represents the variance at Level 2, and σ^2_r, which represents the variance at Level 1 (Hox et al., 2018). For the computation of the intraclass correlation coefficients, see 'Details' in the `multilevel.icc` function. The design effect represents the effect of cluster sampling on the variance of parameter estimation and is defined by the equation

$$deff = (\frac{SE_{Cluster}}{SE_{Simple}})^2 = 1 + \rho(J-1)$$

where $SE_{Cluster}$ is the standard error under cluster sampling, SE_{Simple} is the standard error under simple random sampling, ρ is the intraclass correlation coefficient, ICC(1), and J is the average cluster size. The effective sample size is defined by the equation:

$$N_{effective} = \frac{Ntotal}{deff}$$

The effective sample size $N_{effective}$ represents the equivalent total sample size that we should use in estimating the standard error (Snijders & Bosker, 2012).

Three-Level Model In a three-level model, the intraclass correlation coefficients, design effect, and the effective sample size are computed based on the random intercept-only model:

$$Y_{ijk} = \gamma_{000} + v_{0k} + u_{0jk} + r_{ijk}$$

where the variance in Y is decomposed into three independent components: $\sigma^2_{v_0}$, which represents the variance at Level 3, $\sigma^2_{u_0}$, which represents the variance at Level 2, and σ^2_r, which represents the variance at Level 1 (Hox et al., 2018). For the computation of the intraclass correlation coefficients, see 'Details' in the `multilevel.icc` function. The design effect represents the effect of cluster sampling on the variance of parameter estimation and is defined by the equation

$$deff = (\frac{SE_{Cluster}}{SE_{Simple}})^2 = 1 + \rho_{L2}(J-1) + \rho_{L3}(JK-1)$$

where ρ_{L2} is the ICC(1) at Level 2, ρ_{L3} is the ICC(1) at Level 3, J is the average cluster size at Level 2, and K is the average cluster size at Level 3.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data` including the cluster variable(s) specified in `cluster`
`args`	specification of function arguments
`model.fit`	fitted lavaan object (`mod.fit`)
`result`	list with result tables, i.e., `no.obs` for the number of observations, `no.no.miss` for the number of missing value, `no.cluster.l2` and `no.cluster.l3` for the number of clusters at Level 2 and/or Level 3, `m.cluster.size.l2` and `m.cluster.size.l3` for the average cluster size at Level 2 and/or Level 3, `sd.cluster.size.l2`

and sd.cluster.size.l3 for the standard deviation of the cluster size at Level 2 and/or Level 3, min.cluster.size.l2 min.cluster.size.l3 for the minimum cluster size at Level 2 and/or Level 3, max.cluster.size.l2 max.cluster.size.l3 for the maximum cluster size at Level 2 and/or Level 3, mean.x for the intercept of the multilevel model, var.r for the variance within clusters, var.u for the variance between Level 2 clusters, var.b for the variance between Level 3 clusters, icc1.l2 and icc1.l3 for ICC(1) at Level 2 and/or Level 3, icc2.l2 and icc2.l3 for ICC(2) at Level 2 and/or Level 3, deff for the design effect, deff.sqrt for the square root of the design effect, n.effect for the effective sample size

Author(s)

Takuya Yanagida

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

write.result, multilevel.icc, descript

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Two-Level Data

#————————————————————————————————————
## Cluster variable specification

# Example 1a: Specification using the argument '...'
multilevel.descript(Demo.twolevel, y1, cluster = "cluster")

# Example 1b: Alternative specification with  cluster variable 'cluster' in 'data'
multilevel.descript(Demo.twolevel[, c("y1", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.descript(Demo.twolevel$y1, cluster = Demo.twolevel$cluster)

#————————————————————————————————————
## Arguments 'print', 'icc.digits', 'method'
```

```
# Example 2a: Multilevel descriptive statistics, print variance and standard deviation
multilevel.descript(Demo.twolevel, y1, cluster = "cluster", print = "all")

# Example 2b: Multilevel descriptive statistics, print ICC with 5 digits
multilevel.descript(Demo.twolevel, y1, cluster = "cluster", icc.digits = 5)

# Example 2c: Multilevel descriptive statistics
# use lme() function in the nlme package to estimate ICC
multilevel.descript(Demo.twolevel, y1, cluster = "cluster", method = "nlme")

# Example 2d: Multilevel descriptive statistics for 'y1', 'y2', 'y3', 'w1', and 'w2'
multilevel.descript(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster")

# Alternative specification without using the '...' argument
multilevel.descript(Demo.twolevel[, c("y1", "y2", "y3", "w1", "w2")],
                    cluster = Demo.twolevel$cluster)

#————————————————————————————————————————————————————————————————————————
# Three-Level Data

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

#————————————————————————————————————
## Cluster Variable Specification

# Example 3a: Specification using the argument '...'
multilevel.descript(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"))

# Example 3b: Alternative specification without using the argument '...'
multilevel.descript(Demo.threelevel[, c("y1", "cluster3", "cluster2")],
                    cluster = c("cluster3", "cluster2"))

# Example 3c: Alternative specification with cluster variables 'cluster' not in 'data'
multilevel.descript(Demo.threelevel$y1,
                    cluster = Demo.threelevel[, c("cluster3", "cluster2")])

# Example 3: Multilevel descriptive statistics for 'y1', 'y2', 'y3', 'w1', and 'w2'
multilevel.descript(Demo.threelevel, y1:y3, w1, w2, cluster = c("cluster3", "cluster2"))

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 9a: Write Results into a text file
multilevel.descript(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster",
                    write = "Multilevel_Descript.txt")

# Example 9b: Write Results into a Excel file
multilevel.descript(Demo.twolevel, y1, y2, y3, w1, w2, cluster = "cluster",
                    write = "Multilevel_Descript.xlsx")

## End(Not run)
```

`multilevel.fit` *Simultaneous and Level-Specific Multilevel Model Fit Information*

Description

This function provides simultaneous and level-specific model fit information using the partially saturated model method for multilevel models estimated with the **lavaan** package. Note that level-specific fit indices cannot be computed when the fitted model contains cross-level constraints, e.g., equal factor loadings across levels in line with the metric cross-level measurement invariance assumption.

Usage

```
multilevel.fit(model, print = c("all", "summary", "fit"), digits = 3, p.digits = 3,
               write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted multilevel model of class `"lavaan"` from the **lavaan** package.
`print`	a character vector indicating which results to show on the console, i.e. `"all"` for all results, `"summary"` for a summary of the specification of the estimation method and missing data handling in lavaan and `"fit"` for model fit. By default, the function only prints simultaneous and level-specific multilevel model fit information.
`digits`	an integer value indicating the number of decimal places to be used for displaying results. Note that loglikelihood, information criteria and chi-square test statistic is printed with `digits` minus 1 decimal places.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`args`	specification of function arguments

`model.fit`	a fitted model of class "lavaan"
`result`	list with result tables, i.e., `summary` for the summary of the specification of the estimation method and missing data handling in lavaan and `fit` for the model fit information.

Note

The function uses the functions `fitmeasures`, `lavInspect`, `lavTech`, and `parTable` provided in the R package **lavaan** by Yves Rosseel (2012).

Author(s)

Takuya Yanagida

References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

See Also

`multilevel.cfa`, `multilevel.invar`, `multilevel.omega`, `multilevel.cor`, `multilevel.descript`

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————
# Model Specification and Model Estimation

# Load lavaan package
library(lavaan)

# Model specification
model <- 'level: 1
              fw =~ y1 + y2 + y3
              fw ~ x1 + x2 + x3
           level: 2
              fb =~ y1 + y2 + y3
              fb ~ w1 + w2'

# Model estimation with estimator = "ML"
fit1 <- sem(model = model, data = Demo.twolevel, cluster = "cluster", estimator = "ML")

# Model estimation with estimator = "MLR"
fit2 <- sem(model = model, data = Demo.twolevel, cluster = "cluster", estimator = "MLR")

#————————————————————————————————————————————————————————————————————
```

```
# Simultaneous and Level-Specific Multilevel Model Fit Information

# Example 1a: Standard fit indices
multilevel.fit(fit1)

# Example 1b: Standard, scaled, and robust fit indices
multilevel.fit(fit2)

#-----------------------------------------------------------------------
# Write Results

# Example 2a: Write results into a text file
multilevel.fit(fit1, write = "LS-Fit1.txt")

# Example 2b: Write results into an Excel file
multilevel.fit(fit2, write = "LS-Fit2.xlsx")

## End(Not run)
```

`multilevel.icc` *Intraclass Correlation Coefficient, ICC(1) and ICC(2)*

Description

This function computes the intraclass correlation coefficient ICC(1), i.e., proportion of the total variance explained by the grouping structure, and ICC(2), i.e., reliability of aggregated variables in a two-level and three-level model.

Usage

```
multilevel.icc(data, ..., cluster, type = c("1a", "1b", "2"),
               method = c("lme4", "nlme"), REML = TRUE,
               as.na = NULL, check = TRUE)
```

Arguments

`data`	a numeric vector or data frame.
`...`	an expression indicating the variable names in `data`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`cluster`	a character string indicating the name of the cluster variable in `data` for two-level data, a character vector indicating the names of the cluster variables in `data` for three-level data, or a vector or data frame representing the nested grouping structure (i.e., group or cluster variables). Alternatively, a character vector indicating the variable name(s) of the cluster variable(s) in `data`. Note that the cluster variable at Level 3 come first in a three-level model, i.e., `cluster = c("level3", "level2")`.

`type` a character string indicating the type of intraclass correlation coefficient, i.e., `type = "1a"` (default) for ICC(1) and `type = "2"` for ICC(2) when specifying a two-level model (i.e., one cluster variable), and `type = "1a"` (default) for ICC(1) representing the proportion of variance at Level 2 and Level 3, `type = "1b"` representing an estimate of the expected correlation between two randomly chosen elements in the same group, and `type = "2"` for ICC(2) when specifying a three-level model (i.e., two cluster variables). See 'Details' for the formula used in this function.

`method` a character string indicating the method used to estimate intraclass correlation coefficients, i.e., `method = "lme4"` (default) ICC estimated using the `lmer` function in the **lme4** package, `method = "nlme"` ICC estimated using the `lme` function in the **nlme** package.

`REML` logical: if `TRUE` (default), restricted maximum likelihood is used to estimate the null model when using the `lmer` function in the **lme4** package or the `lme` function in the **nlme** package.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x` but not to `cluster`.

`check` logical: if `TRUE` (default), argument specification is checked.

Details

Two-Level Model In a two-level model, the intraclass correlation coefficients are computed in the random intercept-only model:

$$Y_{ij} = \gamma_{00} + u_{0j} + r_{ij}$$

where the variance in Y is decomposed into two independent components: $\sigma^2_{u_0}$, which represents the variance at Level 2, and σ^2_r, which represents the variance at Level 1 (Hox et al., 2018). These two variances sum up to the total variance and are referred to as variance components. The intraclass correlation coefficient, ICC(1) ρ requested by `type = "1a"` represents the proportion of the total variance explained by the grouping structure and is defined by the equation

$$\rho = \frac{\sigma^2_{u_0}}{\sigma^2_{u_0} + \sigma^2_r}$$

The intraclass correlation coefficient, ICC(2) λ_j requested by `type = "2"` represents the reliability of aggregated variables and is defined by the equation

$$\lambda_j = \frac{\sigma^2_{u_0}}{\sigma^2_{u_0} + \frac{\sigma^2_r}{n_j}} = \frac{n_j \rho}{1 + (n_j - 1)\rho}$$

where n_j is the average group size (Snijders & Bosker, 2012).

Three-Level Model In a three-level model, the intraclass correlation coefficients are computed in the random intercept-only model:

$$Y_{ijk} = \gamma_{000} + v_{0k} + u_{0jk} + r_{ijk}$$

where the variance in Y is decomposed into three independent components: $\sigma^2_{v_0}$, which represents the variance at Level 3, $\sigma^2_{u_0}$, which represents the variance at Level 2, and σ^2_r, which represents the variance at Level 1 (Hox et al., 2018). There are two ways to compute the intraclass correlation coefficient in a three-level model. The first method requested by `type = "1a"` represents the proportion of variance at Level 2 and Level 3 and should be used if we are interested in a decomposition of the variance across levels. The intraclass correlation coefficient, ICC(1) ρ_{L2} at Level 2 is defined as:

$$\rho_{L2} = \frac{\sigma^2_{u_0}}{\sigma^2_{v_0} + \sigma^2_{u_0} + \sigma^2_r}$$

The ICC(1) ρ_{L3} at Level 3 is defined as:

$$\rho_{L3} = \frac{\sigma^2_{v_0}}{\sigma^2_{v_0} + \sigma^2_{u_0} + \sigma^2_r}$$

The second method requested by `type = "1b"` represents the expected correlation between two randomly chosen elements in the same group. The intraclass correlation coefficient, ICC(1) ρ_{L2} at Level 2 is defined as:

$$\rho_{L2} = \frac{\sigma^2_{v_0} + \sigma^2_{u_0}}{\sigma^2_{v_0} + \sigma^2_{u_0} + \sigma^2_r}$$

The ICC(1) $\rho_L 3$ at Level 3 is defined as:

$$\rho_{L3} = \frac{\sigma^2_{v_0}}{\sigma^2_{v_0} + \sigma^2_{u_0} + \sigma^2_r}$$

Note that both formula are correct, but express different aspects of the data, which happen to coincide when there are only two levels (Hox et al., 2018).

The intraclass correlation coefficients, ICC(2) requested by `type = "2"` represent the reliability of aggregated variables at Level 2 and Level 3. The ICC(2) λ_j at Level 2 is defined as:

$$\lambda_j = \frac{\sigma^2_{u_0}}{\sigma^2_{u_0} + \frac{\sigma^2_r}{n_j}}$$

The ICC(2) λ_k at Level 3 is defined as:

$$\lambda_k = \frac{\sigma^2_{v_0}}{\frac{\sigma^2_{v_0} + \sigma^2_{u_0}}{n_j} + \frac{\sigma^2_r}{n_k \cdot n_j}}$$

where n_j is the average group size at Level 2 and n_j is the average group size at Level 3 (Hox et al., 2018).

Value

Returns a numeric vector or matrix with intraclass correlation coefficient(s). In a three level model, the label `L2` is used for ICCs at Level 2 and `L3` for ICCs at Level 3.

Author(s)

Takuya Yanagida

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

multilevel.cfa, multilevel.cor, multilevel.descript

Examples

```
# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————
# Two-Level Data

#————————————————————————————————————
## Cluster Variable Specification

# Example 1a: Specification using the argument '...'
multilevel.icc(Demo.twolevel, y1, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.icc(Demo.twolevel[, c("y1", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.icc(Demo.twolevel$y1, cluster = Demo.twolevel$cluster)

#————————————————————————————————————
## Arguments 'print', 'icc.digits', 'method'

# Example 2a: ICC(2)
multilevel.icc(Demo.twolevel, y1, cluster = "cluster", type = "2")

# Example 2b: ICC(1)
# use lme() function in the nlme package to estimate ICC
multilevel.icc(Demo.twolevel, y1, cluster = "cluster", method = "nlme")

# Example 2c: ICC(1) for 'y1', 'y2', and 'y3'
multilevel.icc(Demo.twolevel, y1, y2, y3, cluster = "cluster")

# Alternative specification without using the '...' argument
multilevel.icc(Demo.twolevel[, c("y1", "y2", "y3")], cluster = Demo.twolevel$cluster)

#————————————————————————————————————————————————————————————————————
```

```
# Three-Level Data

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

#————————————————————————————————————
## Cluster Variable Specification

# Example 3a: Specification using the argument '...'
multilevel.icc(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"))

# Example 3b: Alternative specification without using the argument '...'
multilevel.icc(Demo.threelevel[, c("y1", "cluster3", "cluster2")],
               cluster = c("cluster3", "cluster2"))

# Example 3c: Alternative specification with cluster variables 'cluster' not in 'data'
multilevel.icc(Demo.threelevel$y1, cluster = Demo.threelevel[, c("cluster3", "cluster2")])

#-----------------------------------------------------------------------------

# Example 4a: ICC(1), proportion of variance at Level 2 and Level 3
multilevel.icc(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"))

# Example 4b: ICC(1), expected correlation between two randomly chosen elements
# in the same group
multilevel.icc(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "1b")

# Example 4c: ICC(2)
multilevel.icc(Demo.threelevel, y1, cluster = c("cluster3", "cluster2"), type = "2")
```

`multilevel.indirect` *Confidence Interval for the Indirect Effect in a 1-1-1 Multilevel Mediation Model*

Description

This function computes the confidence interval for the indirect effect in a 1-1-1 multilevel mediation model with random slopes based on the Monte Carlo method.

Usage

```
multilevel.indirect(a, b, se.a, se.b, cov.ab = 0, cov.rand, se.cov.rand,
                  nrep = 100000, alternative = c("two.sided", "less", "greater"),
                     seed = NULL, conf.level = 0.95, digits = 3, write = NULL,
                     append = TRUE, check = TRUE, output = TRUE)
```

Arguments

a	a numeric value indicating the coefficient a, i.e., average effect of X on M on the cluster or between-group level.
b	a numeric value indicating the coefficient b, i.e., average effect of M on Y adjusted for X on the cluster or between-group level.
se.a	a positive numeric value indicating the standard error of a.
se.b	a positive numeric value indicating the standard error of b.
cov.ab	a positive numeric value indicating the covariance between a and b.
cov.rand	a positive numeric value indicating the covariance between the random slopes for a and b.
se.cov.rand	a positive numeric value indicating the standard error of the covariance between the random slopes for a and b.
nrep	an integer value indicating the number of Monte Carlo repetitions.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
seed	a numeric value specifying the seed of the random number generator when using the Monte Carlo method.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
digits	an integer value indicating the number of decimal places to be used for displaying
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

In statistical mediation analysis (MacKinnon & Tofighi, 2013), the indirect effect refers to the effect of the independent variable X on the outcome variable Y transmitted by the mediator variable M. The magnitude of the indirect effect ab is quantified by the product of the the coefficient a (i.e., effect of X on M) and the coefficient b (i.e., effect of M on Y adjusted for X). However, mediation in the context of a 1-1-1 multilevel mediation model where variables X, M, and Y are measured at level 1, the coefficients a and b can vary across level-2 units (i.e., random slope). As a result, a and b may covary so that the estimate of the indirect effect is no longer simply the product of the coefficients $\hat{a}\hat{b}$, but $\hat{a}\hat{b} + \tau_{a,b}$, where $\tau_{a,b}$ (i.e., cov.rand) is the level-2 covariance between the random slopes a and b. The covariance term needs to be added to $\hat{a}\hat{b}$ only when random slopes are estimated for both a and b. Otherwise, the simple product is sufficient to quantify the indirect effect, and the indirect function can be used instead.

In practice, researchers are often interested in confidence limit estimation for the indirect effect. There are several methods for computing a confidence interval for the indirect effect in a single-level mediation models (see indirect function). The Monte Carlo (MC) method (MacKinnon et

al., 2004) is a promising method in single-level mediation model which was also adapted to the multilevel mediation model (Bauer, Preacher & Gil, 2006). This method requires seven pieces of information available from the results of a multilevel mediation model:

a Coefficient a, i.e., average effect of X on M on the cluster or between-group level. In Mplus, `Estimate` of the random slope a under `Means` at the `Between Level`.

b Coefficient b, i.e., average effect of M on Y on the cluster or between-group level. In Mplus, `Estimate` of the random slope b under `Means` at the `Between Level`.

se.a Standard error of `a`. In Mplus, `S.E.` of the random slope a under `Means` at the `Between Level`.

se.b Standard error of `b`. In Mplus, `S.E.` of the random slope b under `Means` at the `Between Level`.

cov.ab Covariance between a and b. In Mplus, the estimated covariance matrix for the parameter estimates (i.e., asymptotic covariance matrix) need to be requested by specifying `TECH3` along with `TECH1` in the `OUTPUT` section. In the `TECHNICAL 1 OUTPUT` under `PARAMETER SPECIFICATION FOR BETWEEN`, the numbers of the parameter for the coefficients a and b need to be identified under `ALPHA` to look up `cov.av` in the corresponding row and column in the `TECHNICAL 3 OUTPUT` under `ESTIMATED COVARIANCE MATRIX FOR PARAMETER ESTIMATES`.

cov.rand Covariance between the random slopes for a and b. In Mplus, `Estimate` of the covariance a `WITH` b at the `Between Level`.

se.cov.rand Standard error of the covariance between the random slopes for a and b. In Mplus, `S.E.` of the covariance a `WITH` b at the `Between Level`.

Note that all pieces of information except `cov.ab` can be looked up in the standard output of the multilevel mediation model. In order to specify `cov.ab`, the covariance matrix for the parameter estimates (i.e., asymptotic covariance matrix) is required. In practice, `cov.ab` will oftentimes be very small so that `cov.ab` may be set to 0 (i.e., default value) with negligible impact on the results.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	list with the input specified in `a`, `b`, `se.a`, `se.b`, `cov.ab`, `cov.rand`, and `se.cov.rand`
`args`	specification of function arguments
`result`	list with result tables, i.e., `ab` for the simulated `ab` values and `mc` for the estimate of the indirect effect and the confidence interval

Note

The function was adapted from the interactive web tool by Preacher and Selig (2010).

Author(s)

Takuya Yanagida

References

Bauer, D. J., Preacher, K. J., & Gil, K. M. (2006). Conceptualizing and testing random indirect effects and moderated Mediation in multilevel models: New procedures and recommendations. *Psychological Methods, 11*, 142-163. https://doi.org/10.1037/1082-989X.11.2.142

Kenny, D. A., Korchmaros, J. D., & Bolger, N. (2003). Lower level Mediation in multilevel models. *Psychological Methods, 8*, 115-128. https://doi.org/10.1037/1082-989x.8.2.115

MacKinnon, D. P., Lockwood, C. M., & Williams, J. (2004). Confidence limits for the indirect effect: Distribution of the product and resampling methods. *Multivariate Behavioral Research, 39*, 99-128. https://doi.org/10.1207/s15327906mbr3901_4

MacKinnon, D. P., & Tofighi, D. (2013). Statistical mediation analysis. In J. A. Schinka, W. F. Velicer, & I. B. Weiner (Eds.), *Handbook of psychology: Research methods in psychology* (pp. 717-735). John Wiley & Sons, Inc..

Preacher, K. J., & Selig, J. P. (2010). *Monte Carlo method for assessing multilevel Mediation: An interactive tool for creating confidence intervals for indirect effects in 1-1-1 multilevel models* [Computer software]. Available from http://quantpsy.org/.

See Also

indirect

Examples

```
# Example 1: Confidence Interval for the Indirect Effect
multilevel.indirect(a = 0.25, b = 0.20, se.a = 0.11, se.b = 0.13,
                    cov.ab = 0.01, cov.rand = 0.40, se.cov.rand = 0.02)

# Example 2: Save results of the Monte Carlo method
ab <- multilevel.indirect(a = 0.25, b = 0.20, se.a = 0.11, se.b = 0.13,
                          cov.ab = 0.01, cov.rand = 0.40, se.cov.rand = 0.02,
                          output = FALSE)$result$ab

# Histogram of the distribution of the indirect effect
hist(ab)

## Not run:
# Example 3: Write results into a text file
multilevel.indirect(a = 0.25, b = 0.20, se.a = 0.11, se.b = 0.13,
                    cov.ab = 0.01, cov.rand = 0.40, se.cov.rand = 0.02,
                    write = "ML-Indirect.txt")
## End(Not run)
```

---

multilevel.invar *Cross-Level Measurement Invariance Evaluation*

Description

This function evaluates configural, metric, and scalar cross-level measurement invariance using multilevel confirmatory factor analysis with continuous indicators by calling the `cfa` function in the R package **lavaan**. By default, the function evaluates configural and metric cross-level measurement invariance by providing a table with model fit information (i.e., chi-square test, fit indices and information criteria), and model comparison (i.e., chi-square difference test, change in fit indices, and change in information criteria). Additionally, variance-covariance coverage of the data, descriptive statistics, parameter estimates, modification indices, residual correlation matrix, and relative Opdyke distribution percentile matrix can be requested by specifying the argument `print`.

Usage

```
multilevel.invar(data, ..., cluster, model = NULL, rescov = NULL,
                 invar = c("config", "metric", "scalar"), fix.resid = NULL,
               ident = c("marker", "var", "effect"), estimator = c("ML", "MLR"),
              optim.method = c("nlminb", "em"), missing = c("listwise", "fiml"),
               print = c("all", "summary", "coverage", "descript", "fit", "est",
                           "modind", "resid", "opdyke"),
                 print.fit = c("all", "standard", "scaled", "robust"),
                 mod.minval = 6.63, resid.minval = 0.1,
                 opdyke.prec = 1, opdyke.minmax = c(0.40, 0.60),
                 color = "b.red", style = c("regular", "bold", "italic"),
                 digits = 3, p.digits = 3, as.na = NULL, write = NULL,
                 append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data` a data frame. If `model` is `NULL`, multilevel confirmatory factor analysis based on a measurement model with one factor at the Within and Between level comprising all variables in the data frame is conducted to evaluate cross-level measurement invariance. Note that the cluster variable specified in `cluster` is excluded from `data` when specifying the argument `cluster` using the variable name of the cluster variable. If `model` is specified, the data frame needs to contain all variables used in the `model` argument.

`...` an expression indicating the variable names in `data`, e.g., `multilevel.invar(dat, x1, x2, x3, cluster = "cluster")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`cluster` either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).

`model` a character vector specifying the same factor structure with one factor at the Within and Between Level, or a list of character vectors for specifying the same measurement model with more than one factor at the Within and Between Level, e.g.,`model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `wf` at the Within level and a measurement model with one factor labeled `bf` at the Between level each comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"),factor2 = c("x5",`

"x6", "x7", "x8")) for specifying a measurement model with two latent factors labeled wfactor1 and wfactor2 at the Within level and a measurement model with two latent factors labeled bfactor1 and bfactor2 at the Between level each comprising four indicators. Note that the name of each list element is used to label factors, where prefixes w and b are added the labels to distinguish factor labels at the Within and Between level, i.e., all list elements need to be named, otherwise factors are labeled with "wf1", "wf2", "wf3" for labels at the Within level and "bf1", "bf2", "bf3" for labels at the Between level and so on.

rescov — a character vector or a list of character vectors for specifying residual covariances at the Within level, e.g. rescov = c("x1", "x2") for specifying a residual covariance between indicators x1 and x2 at the Within level or rescov = list(c("x1", "x2"), c("x3", "x4")) for specifying residual covariances between indicators x1 and x2, and indicators x3 and x4 at the Within level. Note that residual covariances at the Between level can only be specified by using the arguments model.w, model.b, and model.b.

invar — a character string indicating the level of measurement invariance to be evaluated, i.e., config to evaluate configural measurement invariance (i.e., same factor structure across levels), metric (default) to evaluate configural and metric measurement invariance (i.e., equal factor loadings across level), and scalar to evaluate configural, metric and scalar measurement invariance (i.e., all residual variances at the Between level equal zero).

fix.resid — a character vector for specifying residual variances to be fixed at 0 at the Between level for the configural and metric invariance model, e.g., fix.resid = c("x1", "x3") to fix residual variances of indicators x1 and x2 at the Between level at 0. Note that it is also possible to specify fix.resid = "all" which fixes all residual variances at the Between level at 0 in line with the strong factorial measurement invariance assumption across cluster.

ident — a character string indicating the method used for identifying and scaling latent variables, i.e., "marker" for the marker variable method fixing the first factor loading of each latent variable to 1, "var" for the fixed variance method fixing the variance of each latent variable to 1, or "effect" for the effects-coding method using equality constraints so that the average of the factor loading for each latent variable equals 1.

estimator — a character string indicating the estimator to be used: "ML" for maximum likelihood with conventional standard errors and "MLR" (default) for maximum likelihood with Huber-White robust standard errors and a scaled test statistic that is asymptotically equal to the Yuan-Bentler test statistic. Note that by default, full information maximum likelihood (FIML) method is used to deal with missing data when using "ML" (missing = "fiml"), whereas incomplete cases are removed listwise (i.e., missing = "listwise") when using "MLR".

optim.method — a character string indicating the optimizer, i.e., "nlminb" (default) for the unconstrained and bounds-constrained quasi-Newton method optimizer and "em" for the Expectation Maximization (EM) algorithm.

missing — a character string indicating how to deal with missing data, i.e., "listwise" (default) for listwise deletion or "fiml" for full information maximum likelihood (FIML) method. Note that FIML method is only available when estimator =

"ML", that it takes longer to estimate the model using FIML, and that FIML is prone to convergence issues which might be resolved by switching to listwise deletion.

print — a character vector indicating which results to show on the console, i.e. "all" for all results, "summary" for a summary of the specification of the estimation method and missing data handling in lavaan, "coverage" for the variance-covariance coverage of the data, "descript" for descriptive statistics, "fit" for model fit and model comparison, "est" for parameter estimates, and "modind" for modification indices. By default, a summary of the specification and model fit and model comparison are printed.

print.fit — a character vector indicating which version of the CFI, TLI, and RMSEA to show on the console, i.e., "all" for all versions of the CFI, TLI, and RMSEA, "standard" (default when estimator = "ML") for fit indices without any non-normality correction, "scaled" for population-corrected robust fit indices with ad hoc non-normality correction, and robust (default when estimator = "MLR") for sample-corrected robust fit indices based on formula provided by Li and Bentler (2006) and Brosseau-Liard and Savalei (2014).

mod.minval — numeric value to filter modification indices and only show modifications with a modification index value equal or higher than this minimum value. By default, modification indices equal or higher 6.63 are printed. Note that a modification index value of 6.63 is equivalent to a significance level of $\alpha = .01$.

resid.minval — numeric value indicating the minimum absolute residual correlation coefficients and standardized means to highlight in boldface. By default, absolute residual correlation coefficients and standardized means equal or higher 0.1 are highlighted. Note that highlighting can be disabled by setting the minimum value to 1.

opdyke.prec — a numeric value indicating the precision of the probability density function calculations of the Opdyke distribution. The default is 1 which calculates the PDF for polar angles between $(0, pi)$ in 0.01 increments. Specifying 10 calculates the PDF polar angles between $(0, pi)$ in 0.001 increments, which takes considerably longer, especially if there are many correlation elements.

opdyke.minmax — a numeric vector with two elements indicating the minimum and maximum percentile of the Opdyke distribution that is considered to be acceptably close to the observed correlation represented by the Opdyke distribution median. Predicted correlation outside the range will be color highlighted in line with to the argument color.

color — a character string indicating the text color for highlighting absolute residual correlation coefficients and standardized means equal or higher resid.minval and predicted correlations outside the minimum and maximum percentile of the Opdyke distribution, i.e., i.e., "default" for the default text color without color coding and various text colors for highlighting "red", "b.red" (default), "green", "b.green", "blue", or "b.blue", see the help page of the chr.color function. Note that this option is only supported on the RStudio platform, i.e., the argument switches to "default" when using R Markdown and when writing the output into a text file (.txt).

style — a character vector indicating the font style for highlighting absolute residual correlation coefficients and standardized means equal or higher resid.minval,

i.e., "regular" (default) for regular text, "bold" for bold text, and "italic" for italic text. Note that the font style "bold" and "italic" can be combined, i.e., style = c("bold", "italic") provides a bold and italic text. Note that the argument color needs to be specified to change the style of the text, e.g. color = "black" and style = "bold" to for bold text.

digits — an integer value indicating the number of decimal places to be used for displaying results. Note that information criteria and chi-square test statistic is printed with digits minus 1 decimal places.

p.digits — an integer value indicating the number of decimal places to be used for displaying the *p*-value.

as.na — a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to data but not to cluster.

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

append — logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.

check — logical: if TRUE (default), argument specification, convergence and model identification is checked.

output — logical: if TRUE (default), output is shown.

Value

Returns an object of class misty.object, which is a list with following entries:

call — function call

type — type of analysis

data — data frame specified in data

args — specification of function arguments

model — list with specified model for the configural, metric, and scalar invariance model

model.fit — list with fitted lavaan object of the configural, metric, and scalar invariance model

check — list with the results of the convergence and model identification check for the configural, metric, and scalar invariance model

result — list with result tables, i.e., summary for the summary of the specification of the estimation method and missing data handling in lavaan, coverage for the variance-covariance coverage of the data, descript for descriptive statistics, fit for a list with model fit based on standard, scaled, and robust fit indices, est for a list with parameter estimates for the configural, metric, and scalar invariance model, and modind for the list with modification indices for the configural, metric, and scalar invariance model

Note

The function uses the functions `lavTestLRT` provided in the R package **lavaan** by Yves Rosseel (2012). The relative Opdyke distribution percentile matrix is based on the functions `opdyke` and `opdyke.percentiles` provided in the R package **opddyke** by Dan McNeish (2026).

Author(s)

Takuya Yanagida

References

McNeish D (2026). opdyke: Opdyke Percentiles for Approximate Local Fit in Structural Equation Models. R package version 1.0.0. Retrieved from https://github.com/dmcneish18/opdyke

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

See Also

`multilevel.cfa`, `multilevel.fit`, `multilevel.omega`, `multilevel.cor`, `multilevel.descript`

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Cluster Variable Specification

# Example 1a: Specification using the argument '...'
multilevel.invar(Demo.twolevel, y1:y4, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.invar(Demo.twolevel[, c("y1", "y2", "y3", "y4", "cluster")], cluster = "cluster")

# Example 1c: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.invar(Demo.twolevel[, c("y1", "y2", "y3", "y4")], cluster = Demo.twolevel$cluster)

#————————————————————————————————————————————————————————————————————————
# Model Specification Using 'data' for an One-Factor Model

#————————————————————————————————————
## Level of Measurement Invariance

# Example 2a: Configural invariance
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", invar = "config")

# Example 2b: Metric invariance
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", invar = "metric")
```

```
# Example 2c: Scalar invariance
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", invar = "scalar")

#————————————————————————————————————
## Residual (Co-)Variance at the Within and Between Level

# Example 3a: Residual covariance between "y3" and "y4" at the Within level
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster",
                 rescov = c("y3", "y4"))

# Example 3b: Residual variances of 'y1' at the Between level fixed at 0
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", fix.resid = "y1")

#————————————————————————————————————
## Arguments 'print', 'mod.minval', 'resid.minval', and 'opdyke.minmax'

# Example 4a: Request all results
multilevel.invar(Demo.twolevel, y1:y6, cluster = "cluster", print = "all")

# Example 4b: Request modification indices with value equal or higher than 2
multilevel.invar(Demo.twolevel, y1:y6, cluster = "cluster", print = "modind",
                 mod.minval = 2)

# Example 4c: Highlight absolute residual correlation equal or higher than 0.05
multilevel.invar(Demo.twolevel, y1:y6, cluster = "cluster", print = "resid",
                 resid.minval = 0.05, color = "b.blue")

# Example 4d: Highlight Opdyke distribution percentiles outside 0.45 and 0.55
multilevel.invar(Demo.twolevel, y1:y6, cluster = "cluster", print = "opdyke",
                 opdyke.minmax = c(0.45, 0.55), color = "black", style = "bold")

#————————————————————————————————————————————————————————————————————
# lavaan Summary of the Estimated Model

# Example 5: lavaan model and summary of the estimated model
mod <- multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", output = FALSE)

# lavaan syntax of the metric invariance model
mod$model$metric

# Fitted lavaan object of the metric invariance model
lavaan::summary(mod$model.fit$metric, standardized = TRUE, fit.measures = TRUE)

#————————————————————————————————————————————————————————————————————
# Model Specification Using 'model' for an One- or Multiple-Factor Model

# Example 6a: One-factor model
multilevel.invar(Demo.twolevel, cluster = "cluster", model = c("y1", "y2", "y3", "y4"))

# Example 6b:  Two-factor model
multilevel.invar(Demo.twolevel, cluster = "cluster",
                 model = list(c("y1", "y2", "y3"), c("y4", "y5", "y6")))
```

```
#═══════════════════════════════════════════════════════════════════════
# Write Results

# Example 7a: Write Results into a Excel file
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", print = "all",
                 write = "Multilevel_Invariance.txt")

# Example 7b:  Write Results into a Excel file
multilevel.invar(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", print = "all",
                 write = "Multilevel_Invariance.xlsx")

## End(Not run)
```

`multilevel.omega` *Multilevel Composite Reliability*

Description

This function computes point estimate and Monte Carlo confidence interval for the multilevel composite reliability defined by Lai (2021) for a (1) within-cluster construct, (2) shared cluster-level construct, and (3) individual and configural cluster construct by calling the `cfa` function in the R package **lavaan**. By default, the function prints level-specific multilevel composite reliability indices for an individual and configural cluster construct with 95% Monte Carlo confidence interval based on Huber-White standard errors.

Usage

```
multilevel.omega(data, ..., cluster, rescov = NULL,
                 const = c("within", "shared", "config"), fix.resid = NULL,
                 se = c("none", "standard", "robust.huber.white"),
               optim.method = c("nlminb", "em"), missing = c("listwise", "fiml"),
                 nrep = 100000, seed = NULL, conf.level = 0.95,
                 print = c("all", "omega", "item"), digits = 2, r.digits = 3,
                 as.na = NULL, write = NULL, append = TRUE, check = TRUE,
                 output = TRUE)
```

Arguments

`data` a data frame. Multilevel confirmatory factor analysis based on a measurement model with one factor at the Within level and one factor at the Between level comprising all variables in the data frame is conducted. Note that the cluster variable specified in `cluster` is excluded from `data` when specifying the argument `cluster` using the variable name of the cluster variable.

`...` an expression indicating the variable names in `data`, e.g., `multilevel.omega(dat, x1, x2, x3, cluster = "cluster")`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`cluster` either a character string indicating the variable name of the cluster variable in `data`, or a vector representing the nested grouping structure (i.e., group or cluster variable).

`rescov` a character vector or a list of character vectors for specifying residual covariances at the Within level, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between indicators `x1` and `x2` at the Within level or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between indicators `x1` and `x2`, and indicators `x3` and `x4` at the Within level. Note that residual covariances at the Between level cannot be specified using this function.

`const` a character string indicating the type of construct(s), i.e., `"within"` for within-cluster constructs, `"shared"` for shared cluster-level constructs, and `"config"` (default) for individual and configural constructs.

`fix.resid` a character vector for specifying residual variances to be fixed at 0 at the Between level, e.g., `fix.resid = c("x1", "x3")` to fix residual variances of indicators `x1` and `x2` at the Between level at 0. Note that it is also possible to specify `fix.resid = "all"` which fixes all residual variances at the Between level at 0 in line with the strong factorial measurement invariance assumption across cluster.

`se` a character string indicating the standard errors used for computing Monte Carlo confidence intervals, i.e., `"none"` for no standard errors, `"standard"` for conventional standard error based on inverting the expected observed or first.order information matrix, and `"robust.huber.white"` for the 'MLR' (aka pseudo ML, Huber-White) approach. Note that `se = "none"` saves computation time by not computing standard errors and Monte Carlo confidence intervals.

`optim.method` a character string indicating the optimizer, i.e., `"nlminb"` (default) for the unconstrained and bounds-constrained quasi-Newton method optimizer and `"em"` for the Expectation Maximization (EM) algorithm.

`missing` a character string indicating how to deal with missing data, i.e., `"listwise"` for listwise deletion or `"fiml"` (default) for full information maximum likelihood (FIML) method.

`nrep` an integer value indicating the number of Monte Carlo repetitions for computing confidence intervals.

`seed` a numeric value specifying the seed of the random number generator for computing the Monte Carlo confidence interval.

`conf.level` a numeric value between 0 and 1 indicating the confidence level of the interval.

`print` a character vector indicating which results to show, i.e. `"all"` (default) for all results `"omega"` for the composite reliability omega and `"item"` for item statistics.

`digits` an integer value indicating the number of decimal places to be used for displaying mean, standard deviation, minimum, maximum, skewness, and kurtosis.

`r.digits` an integer value indicating the number of decimal places to be used for displaying multilevel coefficient alpha, ICC(1), and standardized factor loadings.

`as.na` a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `data` but not to `cluster`.

`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification, convergence and model identification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Details

Geldhof et al. (2014) introduced a multilevel confirmatory factor analysis approach for estimating score reliability at each level. This approach uses the estimated factor loadings, factor variances, and residual variances from both the within-level and between-level model to compute the within-level reliability $\tilde{\omega}^w$ and the between-level reliability $\tilde{\omega}^b$ based on McDonald's ω. However, Lai (2021) pointed out two limitations of this approach. First, it does not account for different conceptualizations of constructs in multilevel data, where constructs may have different meaning (Stapleton et al., 2016). Second, the between-level reliability $\tilde{\omega}^b$ in Geldhof et al. (2014) is a measure of reliability of the latent cluster means, which ignores the sampling error in the observed cluster means (see Lüdtke et al., 2011). Consequently, $\tilde{\omega}^b$ substantially overestimates the true reliability of observed between-level composite scores. To address these limitations, Lai (2021) proposed multilevel reliability coefficients tailored for different types of constructs to estimate the reliability of the observed scores at each level, i.e., (1) within-cluster construct, (2) shared cluster-level construct, and (3) individual and configural constructs:

Within-Cluster Construct The latent variable representing the within-cluster construct is only meaningful at the within level. For example, students' popularity among their peers within the same classroom based on sociometric ratings. In this case any variation across classrooms should be irrelevant to the construct. Thus, a saturated between-level model is specified. The reliability for a within-level composite, Z_{ij}^w, measuring a within-cluster construct is

$$\omega^w = \frac{(\sum_{k=1}^p \lambda_k^w)^2 \phi^w}{(\sum_{k=1}^p \lambda_k^w)^2 \phi^w + \mathbf{1}'\boldsymbol{\Theta}^w\mathbf{1}}$$

Note that this formula is essentially the same as the formula for the individual construct of an individual and configural construct, except that the factor loadings are specific to the within level as the between-level model is saturated.

Shared Cluster-Level Construct The latent variable representing the shared construct is only meaningful at the between level. For example, safety of a neighborhood, effectiveness of a teacher, or organizational climate are measured by multiple informants at the within level. In these cases, any variation within clusters should be irrelevant to the construct. Thus, a saturated within-level model is specified. The reliability for a between-level composite, Z_j^b, measuring a shared construct is

$$\omega^b = \frac{(\sum_{k=1}^p \lambda_k^b)^2 \phi^b}{(\sum_{k=1}^p \lambda_k^b)^2 \phi^b + \mathbf{1}'\boldsymbol{\Theta}^b\mathbf{1} + \mathbf{1}'\sum^w\mathbf{1}/\tilde{n}}$$

where $\sum^w$ is the variance covariance matrix of the saturated within-level model and $\tilde{n}$ is the harmonic mean of cluster sizes $\tilde{n} = \frac{1}{\sum_{k=1}^{p} \frac{1}{n_j}}$.

Individual and Configural Construct The individual construct is defined at the within level (e.g., individual student achievement), whereas the configural construct consists of cluster averages of individual constructs (e.g., mean student achievement of a classroom). Note that the factor loadings need to be constrained to be equal across levels, i.e., $\lambda^w = \lambda^b$ so that the latent variables at the within level, η^w, and between level, η^b, are on the same metric.

Individual Construct The reliability for a within-level composite, Z_{ij}^w, measuring an individual construct is

$$\omega^w = \frac{(\sum_{k=1}^{p} \lambda_k)^2 \phi^w}{(\sum_{k=1}^{p} \lambda_k)^2 \phi^w + \mathbf{1'\Theta^w 1}}$$

Configural Construct The reliability for a between-level composite, Z_j^b, measuring a configural construct is

$$\omega^b = \frac{(\sum_{k=1}^{p} \lambda_k)^2 \phi^b}{(\sum_{k=1}^{p} \lambda_k)^2 (\phi^b + \phi^w/\tilde{n}) + \mathbf{1'\Theta^b 1} + \mathbf{1'\Theta^w 1}/\tilde{n}}$$

Composite Measuring an Individual and Configural Construct The reliability for an overall composite, Z_{ij}, measuring an individual construct capturing the population variance of both the within-level and the between-level components of the true score and the errors is

$$\omega^{2l} = \frac{(\sum_{k=1}^{p} \lambda_k)^2 (\phi^w + \phi^b)}{(\sum_{k=1}^{p} \lambda_k)^2 (\phi^w + \phi^b) + \mathbf{1'\Theta^b 1} + \mathbf{1'\Theta^w 1}}$$

Note that ω^{2l} is simply the ratio of true variance to total variance of the observed scores. In practice, ω^{2l} is not the most interesting coefficient because researchers are mainly interested in distinguishing within- and between effects (Castro-Alvarez et al., 2026).

Note that the multilevel factor model assumes p items measuring one latent construct at the within level (within-cluster construct), between level (shared cluster-level construct), or within- and between level (individual and configural construct). Local independence is assumed, $\Theta_j^w = diag[\theta_{11}^w ... \theta_{pp}^w]$ and $\Theta_j^b = diag[\theta_{11}^b ... \theta_{pp}^b]$, but composite reliability can be computed when this assumption is violated by specifying the `rescov` argument. The model assumes equal factor loadings across clusters, $\lambda_j^w = \lambda^w$, and homogeneity of error covariances across cluster, $\mathbf{\Theta}_j^w = \mathbf{\Theta}^w$. In case of an individual and configural construct, the model also assumes cross-level measurement invariance, i.e., $\lambda^w = \lambda^b$.

Value

`call`	function call
`type`	type of analysis
`data`	data frame specified in `data` including the group variable specified in `cluster`
`args`	specification of function arguments
`model`	specified model
`model.fit`	fitted lavaan object (`mod.fit`)

`check`	results of the convergence and model identification check
`result`	list with result tables, i.e., `omega` for the coefficient omega including Monte Carlo confidence interval and `itemstat` for descriptive statistics

Note

The function uses the functions `lavInspect`, `lavTech`, and `lavNames`, provided in the R package **lavaan** by Yves Rosseel (2012). The internal function `.internal.mvrnorm` is a copy of the `mvrnorm` function in the package **MASS** by Venables and Ripley (2002).

Author(s)

Takuya Yanagida

References

Castro-Alvarez, S., Bringmann, L. F., Back, J., & Liu, S. (2026). The many reliabilities of psychological dynamics: An overview of statistical approaches to estimate the internal consistency reliability of intensive longitudinal data. *Psychological Methods, 31*(2), 281–296. https://doi.org/10.1037/met0000778

Geldhof, G. J., Preacher, K. J., & Zyphur, M. J. (2014). Reliability estimation in a multilevel confirmatory factor analysis framework. *Psychological Methods, 19*, 72-91. http://dx.doi.org/10.1037/a0032138

Lai, M. H. C. (2021). Composite reliability of multilevel data: It's about observed scores and construct meanings. *Psychological Methods, 26*(1), 90–102. https://doi.org/10.1037/met0000287

Lüdtke, O., Marsh, H. W., Robitzsch, A., & Trautwein, U. (2011). A 2 2 taxonomy of multilevel latent contextual models: Accuracy bias tradeoffs in full and partial error correction models. *Psychological Methods, 16*, 444-467. http://dx.doi.org/10.1037/a0024376

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

Stapleton, L. M., Yang, J. S., & Hancock, G. R. (2016). Construct meaning in multilevel settings. *Journal of Educational and Behavioral Statistics, 41*(5), 481–520. https://doi.org/10.3102/1076998616646200

Venables, W. N., Ripley, B. D. (2002).*Modern Applied Statistics with S* (4th ed.). Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

See Also

`multilevel.alpha`, `item.omega`, `multilevel.cfa`, `multilevel.fit`, `multilevel.invar`, `multilevel.cor`, `multilevel.descript`

Examples

```
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Cluster Variable Specification
```

```
# Example 1a: Specification using the argument '...'
multilevel.omega(Demo.twolevel, y1:y4, cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' in 'data'
multilevel.omega(Demo.twolevel[, c("y1", "y2", "y3", "y4", "cluster")], cluster = "cluster")

# Example 1b: Alternative specification with cluster variable 'cluster' not in 'data'
multilevel.omega(Demo.twolevel[, c("y1", "y2", "y3", "y4")], cluster = Demo.twolevel$cluster)

#————————————————————————————————————————————————————————————————————————
# Type of Construct

# Example 2a: Within-Cluster Construct
multilevel.omega(Demo.twolevel[, c("y1", "y2", "y3", "y4")],
                 cluster = Demo.twolevel$cluster, const = "within")

# Example 2b: Shared Cluster-Level Construct
multilevel.omega(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", const = "shared")

# Example 2c: Configural Construct
multilevel.omega(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", const = "config")

#————————————————————————————————————————————————————————————————————————
# Residual (Co)Variances

# Example 3a: Residual covariance between "y4" and "y5" at the Within level
multilevel.omega(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", const = "config",
                 rescov = c("y3", "y4"))

# Example 3b: Residual variances of 'y1' at the Between level fixed at 0
multilevel.omega(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster", const = "config",
                 fix.resid = c("y1", "y2"))

#————————————————————————————————————————————————————————————————————————
# Arguments 'se' and 'print'

# Example 4a: No confidence intervals to speed up computation
 multilevel.omega(Demo.twolevel, y1:y4, cluster = "cluster", se = "none")

# Example 4b: Omega and item statistics
multilevel.omega(Demo.twolevel, y1:y4, cluster = "cluster", print = "all")

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 5a: Write results into a text file
multilevel.omega(Demo.twolevel[, c("y1", "y2", "y3", "y4")],
                 cluster = Demo.twolevel$cluster, write = "Multilevel_Omega.txt")

# Example 5b: Write results into a Excel file
multilevel.omega(Demo.twolevel, y1, y2, y3, y4, cluster = "cluster",
                 write = "Multilevel_Omega.xlsx")
```

```
## End(Not run)
```

---

multilevel.r2 *R-Squared Measures for Multilevel and Linear Mixed Effects Models*

Description

This function computes R-squared measures by Raudenbush and Bryk (2002), Snijders and Bosker (1994), Nakagawa and Schielzeth (2013) as extended by Johnson (2014), and Rights and Sterba (2019) for multilevel and linear mixed effects models estimated by using the `lmer()` function in the package **lme4** or `lme()` function in the package **nlme**.

Usage

```
multilevel.r2(model, print = c("all", "RB", "SB", "NS", "RS"), digits = 3,
              plot = FALSE, filename = NULL, width = NA, height = NA, dpi = 600,
               write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`model`	a fitted model of class `"lmerMod"` from the **lme4** package or `"lme"` from the **nlme** package.
`print`	a character vector indicating which R-squared measures to be printed on the console, i.e., `RB` for measures from Raudenbush and Bryk (2002), `SB` for measures from Snijders and Bosker (1994), `NS` for measures from Nakagawa and Schielzeth (2013) as extended by Johnson (2014), and `RS` for measures from Rights and Sterba (2019). The default setting is `print = "RS"`.
`digits`	an integer value indicating the number of decimal places to be used.
`plot`	logical: if `TRUE`, bar chart showing the decomposition of scaled total, within-cluster, and between-cluster outcome variance into five (total), three (within-cluster), and two (between-cluster) proportions is drawn. Note that the **ggplot2** package is required to draw the bar chart.
`filename`	a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"` (default), `".jpeg"`, `".tiff"`, `".png"`, `".bmp"`, `".svg"` or `".wmf"` needs to be specified as file extension in the `file` argument. Note that plots can only be saved when `plot = TRUE` and `print = "RS"`.
`width`	a numeric value indicating the `width` argument (default is the size of the current graphics device) in the `ggsave` function.
`height`	a numeric value indicating the `height` argument (default is the size of the current graphics device) in the `ggsave` function.
`dpi`	a numeric value indicating the `dpi` argument (default is `600`) in the `ggsave` function.
`write`	a character string naming a text file with file extension `".txt"` (e.g., `"Output.txt"`) for writing the output into a text file.

`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.
`...`	additional plotting arguments from the `plot.misty.object()` function, see 'Details in the `plot.misty.object` function.

Details

A number of R-squared measures for multilevel and linear mixed effects models have been developed in the methodological literature (see Rights & Sterba, 2018). Based on these measures, following measures were implemented in the current function:

Raudenbush and Bryk (2002) R-squared measures by Raudenbush and Bryk (2002) are based on the proportional reduction of unexplained variance when predictors are added. More specifically, variance estimates from the baseline/null model (i.e., $\sigma^2_{e|b}$ and $\sigma^2_{u0|b}$) and variance estimates from the model including predictors (i.e., $\sigma^2_{e|m}$ and $\sigma^2_{u0|m}$) are used to compute the proportional reduction in variance between baseline/null model and the complete model by:

$$R^2_1(RB) = \frac{\sigma^2_{e|b} - \sigma^2_{e|m}}{\sigma^2_{e|b}}$$

for the proportional reduction at level-1 (within-cluster) and by:

$$R^2_2(RB) = \frac{\sigma^2_{u0|b} - \sigma^2_{u0|m}}{\sigma^2_{u0|b}}$$

for the proportional reduction at level-2 (between-cluster), where $|b$ and $|m$ represent the baseline and full models, respectively (Hox et al., 2018; Roberts et al., 2010).

A major disadvantage of these measures is that adding predictors can increases rather than decreases some of the variance components and it is even possible to obtain negative values for R^2 with these formulas (Snijders & Bosker, 2012). According to Snijders and Bosker (1994) this can occur because the between-group variance is a function of both level-1 and level-2 variance:

$$var(\bar{Y}_j) = \sigma^2_{u0} + \frac{\sigma^2_e}{n_j}$$

Hence, adding a predictor (e.g., cluster-mean centered predictor) that explains proportion of the within-group variance will decrease the estimate of σ^2_e and increase the estimate σ^2_{u0} if this predictor does not explain a proportion of the between-group variance to balance out the decrease in σ^2_e (LaHuis et al., 2014). Negative estimates for R^2 can also simply occur due to chance fluctuation in sample estimates from the two models.

Another disadvantage of these measures is that $R^2_2(RB)$ for the explained variance at level-2 has been shown to perform poorly in simulation studies even with $j = 200$ clusters with group cluster size of $n_j = 50$ (LaHuis et al., 2014; Rights & Sterba, 2019).

Moreover, when there is missing data in the level-1 predictors, it is possible that sample sizes for the baseline and complete models differ.

Finally, it should be noted that R-squared measures by Raudenbush and Bryk (2002) are appropriate for random intercept models, but not for random intercept and slope models. For random slope models, Snijders and Bosker (2012) suggested to re-estimate the model as random intercept models with the same predictors while omitting the random slopes to compute the R-squared measures. However, the simulation study by LaHuis (2014) suggested that the R-squared measures showed an acceptable performance when there was little slope variance, but did not perform well in the presence of higher levels of slope variance.

Snijders and Bosker (1994) R-squared measures by Snijders and Bosker (1994) are based on the proportional reduction of mean squared prediction error and is computed using the formula:

$$R_1^2(SB) = \frac{\hat{\sigma}^2_{e|m} + \hat{\sigma}^2_{u0|m}}{\hat{\sigma}^2_{e|b} + \hat{\sigma}^2_{u0|b}}$$

for computing the proportional reduction of error at level-1 representing the total amount of explained variance and using the formula:

$$R_2^2(SB) = \frac{\hat{\sigma}^2_{e|m}/n_j + \hat{\sigma}^2_{u0|m}}{\hat{\sigma}^2_{e|b}/n_j + \hat{\sigma}^2_{u0|b}}$$

for computing the proportional reduction of error at level-2 by dividing the $\hat{\sigma}^2_e$ by the group cluster size n_j or by the average cluster size for unbalanced data (Roberts et al., 2010). Note that the function uses the harmonic mean of the group sizes as recommended by Snijders and Bosker (1994). The population values of R^2 based on these measures cannot be negative because the interplay of level-1 and level-2 variance components is considered. However, sample estimates of R^2 can be negative either due to chance fluctuation when sample sizes are small or due to model misspecification (Snijders and Bosker, 2012).

When there is missing data in the level-1 predictors, it is possible that sample sizes for the baseline and complete models differ.

Similar to the R-squared measures by Raudenbush and Bryk (2002), the measures by Snijders and Bosker (1994) are appropriate for random intercept models, but not for random intercept and slope models. Accordingly, for random slope models, Snijders and Bosker (2012) suggested to re-estimate the model as random intercept models with the same predictors while omitting the random slopes to compute the R-squared measures. The simulation study by LaHuis et al. (2014) revealed that the R-squared measures showed an acceptable performance, but it should be noted that $R_2^2(SB)$ the explained variance at level-2 was not investigated in their study.

Nakagawa and Schielzeth (2013) R-squared measures by Nakagawa and Schielzeth (2013) are based on partitioning model-implied variance from a single fitted model and uses the variance of predicted values of $var(\hat{Y}_{ij})$ to form both the outcome variance in the denominator and the explained variance in the numerator of the formulas:

$$R_m^2(NS) = \frac{var(\hat{Y}_{ij})}{var(\hat{Y}_{ij}) + \sigma^2_{u0} + \sigma^2_e}$$

for marginal total $R_m^2(NS)$ and:

$$R_c^2(NS) = \frac{var(\hat{Y}_{ij}) + \sigma^2_{u0}}{var(\hat{Y}_{ij}) + \sigma^2_{u0} + \sigma^2_e}$$

for conditional total $R^2_c(NS)$. In the former formula R^2 predicted scores are marginalized across random effects to indicate the variance explained by fixed effects and in the latter formula R^2 predicted scores are conditioned on random effects to indicate the variance explained by fixed and random effects (Rights and Sterba, 2019).

The advantage of these measures is that they can never become negative and that they can also be extended to generalized linear mixed effects models (GLMM) when outcome variables are not continuous (e.g., binary outcome variables). Note that currently the function does not provide R^2 measures for GLMMs, but these measures can be obtained using the `r.squaredGLMM()` function in the **MuMIn** package.

A disadvantage is that these measures do not allow random slopes and are restricted to the simplest random effect structure (i.e., random intercept model). In other words, these measures do not fully reflect the structure of the fitted model when using random intercept and slope models. However, Johnson (2014) extended these measures to allow random slope by taking into account the contribution of random slopes, intercept-slope covariances, and the covariance matrix of random slope to the variance in Y_{ij}. As a result, R-squared measures by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) can be used for both random intercept, and random intercept and slope models.

The major criticism of the R-squared measures by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) is that these measures do not decompose outcome variance into each of total, within-cluster, and between-cluster variance which precludes from computing level-specific R^2 measures. In addition, these measures do not distinguish variance attributable to level-1 versus level-2 predictors via fixed effects, and they also do not distinguish between random intercept and random slope variation (Rights and Sterba, 2019).

Rights and Sterba (2019) R-squared measures by Rights and Sterba (2019) provide an integrative framework of R-squared measures for multilevel and linear mixed effects models with random intercepts and/or slopes. Their measures are also based on partitioning model implied variance from a single fitted model, but they provide a full partitioning of the total outcome variance to one of five specific sources:

- variance attributable to level-1 predictors via fixed slopes (shorthand: variance attributable to `f1`)
- variance attributable to level-2 predictors via fixed slopes (shorthand: variance attributable to `f2`)
- variance attributable to level-1 predictors via random slope variation/ covariation (shorthand: variance attributable to `v`)
- variance attributable to cluster-specific outcome means via random intercept variation (shorthand: variance attributable to `m`)
- variance attributable to level-1 residuals

R^2 measures are based on the outcome variance of interest (total, within-cluster, or between-cluster) in the denominator, and the source contributing to explained variance in the numerator:

Total R^2 measures incorporate both within-cluster and between cluster variance in the denominator and quantify variance explained in an omnibus sense:

- $R^{2(f_1)}_t$: Proportion of total outcome variance explained by level-1 predictors via fixed slopes.
- $R^{2(f_2)}_t$: Proportion of total outcome variance explained by level-2 predictors via fixed slopes.

- $R_t^{2(f)}$: Proportion of total outcome variance explained by all predictors via fixed slopes.
- $R_t^{2(v)}$: Proportion of total outcome variance explained by level-1 predictors via random slope variation/covariation.
- $R_t^{2(m)}$: Proportion of total outcome variance explained by cluster-specific outcome means via random intercept variation.
- $R_t^{2(fv)}$: Proportion of total outcome variance explained by predictors via fixed slopes and random slope variation/covariation.
- $R_t^{2(fvm)}$: Proportion of total outcome variance explained by predictors via fixed slopes and random slope variation/covariation and by cluster-specific outcome means via random intercept variation.

Within-Cluster R^2 measures incorporate only within-cluster variance in the denominator and indicate the degree to which within-cluster variance can be explained by a given model:

- $R_w^{2(f_1)}$: Proportion of within-cluster outcome variance explained by level-1 predictors via fixed slopes.
- $R_w^{2(v)}$: Proportion of within-cluster outcome variance explained by level-1 predictors via random slope variation/covariation.
- $R_w^{2(f_1v)}$: Proportion of within-cluster outcome variance explained by level-1 predictors via fixed slopes and random slope variation/covariation.

Between-Cluster R^2 measures incorporate only between-cluster variance in the denominator and indicate the degree to which between-cluster variance can be explained by a given model:

- $R_b^{2(f_2)}$: Proportion of between-cluster outcome variance explained by level-2 predictors via fixed slopes.
- $R_b^{2(m)}$: Proportion of between-cluster outcome variance explained by cluster-specific outcome means via random intercept variation.

The decomposition of the total outcome variance can be visualized in a bar chart by specifying `plot = TRUE`. The first column of the bar chart decomposes scaled total variance into five distinct proportions (i.e., $R_t^{2(f_1)}$, $R_t^{2(f_2)}$, $R_t^{2(f)}$, $R_t^{2(v)}$, $R_t^{2(m)}$, $R_t^{2(fv)}$, and $R_t^{2(fvm)}$), the second column decomposes scaled within-cluster variance into three distinct proportions (i.e., $R_w^{2(f_1)}$, $R_w^{2(v)}$, and $R_w^{2(f_1v)}$), and the third column decomposes scaled between-cluster variance into two distinct proportions (i.e., $R_b^{2(f_2)}$, $R_b^{2(m)}$).

Note that the function assumes that all level-1 predictors are centered within cluster (i.e., group-mean or cluster-mean centering) as has been widely recommended (e.g., Enders & Tofighi, D., 2007; Rights et al., 2019). In fact, it does not matter whether a lower-level predictor is merely a control variable, or is quantitative or categorical (Yaremych et al., 2021), cluster-mean centering should always be used for lower-level predictors to obtain an orthogonal between-within partitioning of a lower-level predictor's variance that directly parallels what happens to a level-1 outcome (Hoffman & Walters, 2022). In the absence of cluster-mean-centering, however, the function provides total R^2 measures, but does not provide any within-cluster or between-cluster R^2 measures.

By default, the function only computes R-squared measures by Rights and Sterba (2019) because the other R-squared measures reflect the same population quantity provided by Rights and Sterba

(2019). That is, R-squared measures $R_1^2(RB)$ and $R_2^2(RB)$ by Raudenbush and Bryk (2002) are equivalent to $R_w^{2(f_1v)}$ and $R_b^{2(f_2)}$, R-squared measures $R_1^2(SB)$ and $R_2^2(SB)$ are equivalent to $R_t^{2(f)}$ and $R_b^{2(f_2)}$, and R-squared measures $R_m^2(NS)$ and $R_c^2(NS)$ by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) are equivalent to $R_t^{2(f)}$ and $R_t^{2(fvm)}$ (see Rights and Sterba, Table 3).

Note that none of these measures provide an R^2 for the random slope variance explained by cross-level interactions, a quantity that is frequently of interest (Hoffman & Walters, 2022).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	matrix or data frame specified in `data`
`plot`	ggplot2 object for plotting the results
`args`	specification of function arguments
`result`	list with result tables, i.e., `rb` for the R2 measures by Raudenbush and Bryk (2002), `sb` for the R2 measures by Snijders and Bosker (1994), `ns` for the R2 measures by Nakagawa and Schielzeth (2013), and `rs` for the R2 measures by Rights and Sterba (2019)

Note

This function is based on the `multilevelR2()` function from the **mitml** package by Simon Grund, Alexander Robitzsch and Oliver Luedtke (2021), fhe `r.squaredGLMM()` function from the **MuMIn** by Kamil Bartoń, and a copy of the function `r2mlm` in the **r2mlm** package by Mairead Shaw, Jason Rights, Sonya Sterba, and Jessica Flake.

Author(s)

Takuya Yanagida

References

Bartoń K (2025). *MuMIn: Multi-Model Inference*. R package version 1.48.1. https://CRAN.R-project.org/package=MuMIn.

Enders, C. K., & Tofighi, D. (2007). Centering predictor variables in cross-sectional multilevel models: A new look at an old issue. *Psychological Methods, 12*, 121-138. https://doi.org/10.1037/1082-989X.12.2.121

Hoffmann, L., & Walter, W. R. (2022). Catching up on multilevel modeling. *Annual Review of Psychology, 73*, 629-658. https://doi.org/10.1146/annurev-psych-020821-103525

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel Analysis: Techniques and Applications* (3rd ed.) Routledge.

Johnson, P. C. D. (2014). Extension of Nakagawa & Schielzeth's R2 GLMM to random slopes models. *Methods in Ecology and Evolution, 5*(9), 944-946. https://doi.org/10.1111/2041-210X.12225

LaHuis, D. M., Hartman, M. J., Hakoyama, S., & Clark, P. C. (2014). Explained variance measures for multilevel models. *Organizational Research Methods, 17*, 433-451. https://doi.org/10.1177/1094428114541701

Nakagawa, S., & Schielzeth, H. (2013). A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution, 4*(2), 133-142. https://doi.org/10.1111/j.2041-210x.2012.00261.x

Raudenbush, S. W., & Bryk, A. S., (2002). *Hierarchical linear models: Applications and data analysis methods*. Sage.

Rights, J. D., Preacher, K. J., & Cole, D. A. (2020). The danger of conflating level-specific effects of control variables when primary interest lies in level-2 effects. *British Journal of Mathematical and Statistical Psychology, 73*(Suppl 1), 194-211. https://doi.org/10.1111/bmsp.12194

Rights, J. D., & Sterba, S. K. (2019). Quantifying explained variance in multilevel models: An integrative framework for defining R-squared measures. *Psychological Methods, 24*, 309-338. https://doi.org/10.1037/met0000184

Roberts, K. J., Monaco, J. P., Stovall, H., & Foster, V. (2011). Explained variance in multilevel models (pp. 219-230). In J. J. Hox & J. K. Roberts (Eds.), *Handbook of Advanced Multilevel Analysis*. Routledge.

Snijders, T. A. B., & Bosker, R. (1994). Modeled variance in two-level models. *Sociological methods and research, 22*, 342-363. https://doi.org/10.1177/0049124194022003004

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage.

Yaremych, H. E., Preacher, K. J., & Hedeker, D. (2021). Centering categorical predictors in multilevel models: Best practices and interpretation. *Psychological Methods*. Advance online publication. https://doi.org/10.1037/met0000434

See Also

multilevel.cor, multilevel.descript, multilevel.icc, multilevel.indirect

Examples

```
## Not run:

# Load misty, lme4, nlme, and ggplot2 package
misty::libraries(misty, lme4, nlme, ggplot2)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#———————————————————————————————————————————————————————————————————
# Estimate Multilevel Models

# Cluster mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Compute group means, cluster.scores() from the misty package
Demo.twolevel <- cluster.scores(Demo.twolevel, x2, cluster = "cluster", name = "x2.b")
```

```
# Estimate multilevel model using the lme4 package
mod1a <- lmer(y1 ~ x2.c + x2.b + w1 + (1 + x2.c | cluster), data = Demo.twolevel,
              REML = FALSE, control = lmerControl(optimizer = "bobyqa"))

# Estimate multilevel model using the nlme package
mod1b <- lme(y1 ~ x2.c + x2.b + w1, random = ~ 1 + x2.c | cluster, data = Demo.twolevel,
             method = "ML")

#————————————————————————————————————————————————————————————————————————
# R-Squared Measures According to Rights and Sterba (2019)

# Example 1a: Multilevel model using the lmer() function
multilevel.r2(mod1a)

# Example 1b: Multilevel model using the lme() function
multilevel.r2(mod1b)

# Example 1c: Write Results into a text file
multilevel.r2(mod1a, write = "ML-R2.txt")

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 2a: Default setting
multilevel.r2(mod1a, plot = TRUE)

# Example 2b: Bar chart, additional arguments
# see Details in the help page of the function plot.misty.object
multilevel.r2(mod1a, plot = TRUE, gray = TRUE)

# Example 2c: Save bar chart
multilevel.r2(mod1a, plot = TRUE, filename = "Proportion_of_Variance.png",
              dpi = 600, width = 5.5, height = 5.5)

#————————————————————————————————————————————————————————————————————————
# Argument 'print'

# Estimate multilevel model without random slopes
# Note. R-squared measures by Raudenbush and Bryk (2002), and  Snijders and
# Bosker (2012) should be computed based on the random intercept model
mod2 <- lmer(y1 ~ x2.c + x2.b + w1 + (1 | cluster), data = Demo.twolevel,
             REML = FALSE, control = lmerControl(optimizer = "bobyqa"))

# Example 3: Print all available R-squared measures
multilevel.r2(mod2, print = "all")

#————————————————————————————————————————————————————————————————————————
# Draw Bar Chart Manually

# Extract misty object
mod1a.r2 <- multilevel.r2(mod1a, output = FALSE)

# Prepare data frame for ggplot()
```

```
df <- data.frame(var = factor(rep(c("Total", "Within", "Between"), each = 5),
                              level = c("Total", "Within", "Between")),
                 part = factor(c("Fixed Slopes (Within)", "Fixed Slopes (Between)",
                            "Slope Variation (Within)", "Intercept Variation (Between)",
                                "Residual (Within)"),
                 level = c("Residual (Within)", "Intercept Variation (Between)",
                           "Slope Variation (Within)", "Fixed Slopes (Between)",
                           "Fixed Slopes (Within)")),
                 y = as.vector(mod1a.r2$result$rs$decomp))

# Example 4: Draw bar chart in line with the default setting of multilevel.r2()
ggplot(df, aes(x = var, y = y, fill = part)) +
  theme_bw() +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("#E69F00", "#009E73", "#CC79A7", "#0072B2", "#D55E00")) +
  scale_y_continuous(name = "Proportion of Variance", breaks = seq(0, 1, by = 0.1)) +
  theme(axis.title.x = element_blank(),
        axis.ticks.x = element_blank(),
        legend.title = element_blank(),
        legend.position = "bottom",
        legend.box.margin = margin(-10, 6, 6, 6)) +
  guides(fill = guide_legend(nrow = 2, reverse = TRUE))

## End(Not run)
```

`multilevel.r2.manual` *R-Squared Measures for Multilevel and Linear Mixed Effects Models by Rights and Sterba (2019), Manually Inputting Parameter Estimates*

Description

This function computes R-squared measures by Rights and Sterba (2019) for multilevel and linear mixed effects models by manually inputting parameter estimates.

Usage

```
multilevel.r2.manual(data, within = NULL, between = NULL, random = NULL,
                     gamma.w = NULL, gamma.b = NULL, tau, sigma2,
                     intercept = TRUE, center = TRUE, digits = 3,
                     plot = FALSE, filename = NULL, width = NA, height = NA,
                     dpi = 600, write = NULL, append = TRUE, check = TRUE,
                     output = TRUE, ...)
```

Arguments

`data`	a matrix or data frame with the level-1 and level-2 predictors and outcome variable used in the model.
`within`	a character vector with the variable names in `data` or numeric vector with numbers corresponding to the columns in `data` of the level-1 predictors used in the model. If none used, set to `NULL`.

between	a character vector with the variable names in data or numeric vector with numbers corresponding to the columns in data of the level-2 predictors used in the model. If none used, set to NULL.
random	a character vector with the variable names in data or numeric vector with numbers corresponding to the columns in data of the level-1 predictors that have random slopes in the model. If no random slopes specified, set to NULL.
gamma.w	a numeric vector of fixed slope estimates for all level-1 predictors, to be entered in the order of the predictors listed in the argument within.
gamma.b	a numeric vector of the intercept and fixed slope estimates for all level-2predictors, to be entered in the order of the predictors listed in the argument between. Note that the first element is the parameter estimate for the intercept if intercept = TRUE.
tau	a matrix indicating the random effects covariance matrix, the first row/column denotes the intercept variance and covariances (if intercept is fixed, set all to 0) and each subsequent row/column denotes a given random slope's variance and covariances (to be entered in the order listed in the argument random).
sigma2	a numeric value indicating the level-1 residual variance.
intercept	logical: if TRUE (default), the first element in the gamma.b is assumed to be the fixed intercept estimate; if set to FALSE, the first element in the argument gamma.b is assumed to be the first fixed level-2 predictor slope.
center	logical: if TRUE (default), all level-1 predictors are assumed to be cluster-mean-centered and the function will output all decompositions; if set to FALSE, function will output only the total decomposition.
digits	an integer value indicating the number of decimal places to be used.
plot	logical: if TRUE, bar chart showing the decomposition of scaled total, within-cluster, and between-cluster outcome variance into five (total), three (within-cluster), and two (between-cluster) proportions is drawn. Note that the **ggplot2** package is required to draw the bar chart.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) in the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.

output	logical: if TRUE (default), output is shown on the console.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

A number of R-squared measures for multilevel and linear mixed effects models have been developed in the methodological literature (see Rights & Sterba, 2018). R-squared measures by Rights and Sterba (2019) provide an integrative framework of R-squared measures for multilevel and linear mixed effects models with random intercepts and/or slopes. Their measures are based on partitioning model implied variance from a single fitted model, but they provide a full partitioning of the total outcome variance to one of five specific sources. See the help page of the multilevel.r2 function for more details.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	matrix or data frame specified in data
plot	ggplot2 object for plotting the results
args	specification of function arguments
result	list with result tables, i.e., decomp for the decomposition, total for total R2 measures, within for the within-cluster R2 measures, and between

for the between-cluster R2 measures.

Note

This function is based on a copy of the function r2mlm_manual() in the **r2mlm** package by Mairead Shaw, Jason Rights, Sonya Sterba, and Jessica Flake.

Author(s)

Takuya Yanagida

References

Rights, J. D., & Cole, D. A. (2018). Effect size measures for multilevel models in clinical child and adolescent research: New r-squared methods and recommendations. *Journal of Clinical Child and Adolescent Psychology, 47*, 863-873. https://doi.org/10.1080/15374416.2018.1528550

Rights, J. D., & Sterba, S. K. (2019). Quantifying explained variance in multilevel models: An integrative framework for defining R-squared measures. *Psychological Methods, 24*, 309-338. https://doi.org/10.1037/met0000184

See Also

multilevel.r2, multilevel.cor, multilevel.descript, multilevel.icc, multilevel.indirect

Examples

```
## Not run:

# Load misty and lme4 package
misty::libraries(misty, lme4)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#———————————————————————————————————————————————————————————————————
# Estimate Multilevel Models

Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Compute group means, cluster.scores() from the misty package
Demo.twolevel <- cluster.scores(Demo.twolevel, x2, cluster = "cluster", name = "x2.b")

# Estimate random intercept model using the lme4 package
mod1 <- lmer(y1 ~ x2.c + x2.b + w1 + (1| cluster), data = Demo.twolevel,
             REML = FALSE, control = lmerControl(optimizer = "bobyqa"))

# Estimate random intercept and slope model using the lme4 package
mod2 <- lmer(y1 ~ x2.c + x2.b + w1 + (1 + x2.c | cluster), data = Demo.twolevel,
             REML = FALSE, control = lmerControl(optimizer = "bobyqa"))

#———————————————————————————————————————————————————————————————————
# R-Squared Measures According to Rights and Sterba (2019)

# Fixed slope estimates
fixef(mod1)

# Random effects variance-covariance matrix
as.data.frame(VarCorr(mod1))

# Example 1a: R-squared measures for a random intercept model
multilevel.r2.manual(data = Demo.twolevel,
                     within = "x2.c", between = c("x2.b", "w1"),
                     gamma.w = 0.41127956,
                     gamma.b = c(0.01123245, -0.08269374, 0.17688507),
                     tau = 0.9297401,
                     sigma2 = 1.813245794)

#————————————————————————————————————

# Fixed slope estimates
fixef(mod2)

# Random effects variance-covariance matrix
as.data.frame(VarCorr(mod2))

# Example 1b: R-squared measures for a random intercept and slope model
multilevel.r2.manual(data = Demo.twolevel,
```

```
                    within = "x2.c", between = c("x2.b", "w1"), random = "x2.c",
                    gamma.w = 0.41127956,
                    gamma.b = c(0.01123245, -0.08269374, 0.17688507),
              tau = matrix(c(0.931008649, 0.004110479, 0.004110479, 0.017068857), ncol = 2),
                    sigma2 = 1.813245794)

## End(Not run)
```

---

na.as *Replace Missing Values With User-Specified Values or User-Specified Values With Missing Values*

Description

The function na.as replaces NA in a vector, factor, list, matrix or data frame with a user-specified value or character string in the argument na, while the function as.na replaces user-specified values in the argument na in a vector, factor, matrix, array, list, or data frame with NA.

Usage

```
na.as(data, ..., na, replace = TRUE, as.na = NULL, check = TRUE)

as.na(data, ..., na, replace = TRUE, check = TRUE)
```

Arguments

data	a vector, factor, matrix, array, data frame, or list.
...	an expression indicating the variable names in data, e.g., as.na(dat, x1, x2) for selecting the variables x1 and x2 from the data frame dat. Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
na	a vector indicating values or characters to replace with NA, or which NA is replaced. Note that a numeric value or character string needs to be specified for the argument na when using na.as.
replace	logical: if TRUE (default), variable(s) specified in ... are replaced in the argument data.
check	logical: if TRUE (default), argument specification is checked.
as.na	a numeric vector or character vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.

Value

Returns a vector, factor, matrix, array, data frame, or list specified in the argument data.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

na.auxiliary, na.coverage, na.descript, na.indicator, na.pattern, na.prop, na.test

Examples

```
#-----------------------------------------------------------------------
# Numeric Vector

num <- c(1, 3, 2, 4, 5)

# Example 11: Replace NA with 2
na.as(c(1, 3, NA, 4, 5), na = 2)

# Example 1b: Replace 2 with NA
as.na(num, na = 2)

# Example 1c: Replace 2, 3, and 4 with NA
as.na(num, na = c(2, 3, 4))

#-----------------------------------------------------------------------
# Character Vector

chr <- c("a", "b", "c", "d", "e")

# Example 2a: Replace NA with "b"
na.as(c("a", NA, "c", "d", "e"), na = "b")

# Example 2b: Replace "b" with NA
as.na(chr, na = "b")

# Example 2c: Replace "b", "c", and "d" with NA
as.na(chr, na = c("b", "c", "d"))

#-----------------------------------------------------------------------
# Factor

fac <- factor(c("a", "a", "b", "b", "c", "c"))

# Example 3a: Replace NA with "b"
na.as(factor(c("a", "a", NA, NA, "c", "c")), na = "b")

# Example 3b: Replace "b" with NA
as.na(fac, na = "b")

# Example 3c: Replace "b" and "c" with NA
as.na(fac, na = c("b", "c"))
```

```
#──────────────────────────────────────────────────────────────────────────────
# Matrix

mat <- matrix(1:20, ncol = 4)

# Example 4a: Replace NA with 2
na.as(matrix(c(1, NA, 3, 4, 5, 6), ncol = 2), na = 2)

# Example 4b: Replace 8 with NA
as.na(mat, na = 8)

# Example 4c: Replace 8, 14, and 20 with NA
as.na(mat, na = c(8, 14, 20))

#──────────────────────────────────────────────────────────────────────────────
# Array

# Example 5: Replace 1 and 10 with NA
as.na(array(1:20, dim = c(2, 3, 2)), na = c(1, 10))

#──────────────────────────────────────────────────────────────────────────────
# List

# Example 6:  Replace 1 with NA
as.na(list(x1 = c(1, 2, 3, 1, 2, 3), x2 = c(2, 1, 3, 2, 1)), na = 1)

#──────────────────────────────────────────────────────────────────────────────
# Data frame

df <- data.frame(x1 = c(1, NA, 3), x2 = c(2, 1, 3), x3 = c(3, NA, 2))

# Example 7a: Replace NA with -99
na.as(df, na = -99)

# Example 7b: Replace 1 with NA
as.na(df, na = 1)

# Example 7c: Replace 1 with NA for the variable 'x2'
as.na(df, x2, na = 1)

# Alternative specification
as.na(df$x2, na = 1)

# Example 7d: Replace 1 and 3 with NA
as.na(df, na = c(1, 3))

# Example 7e: Replace 1 with NA in 'x2' and 'x3'
as.na(df, x2, x3, na = 1)
```

`na.auxiliary` *Auxiliary Variables Analysis*

Description

This function computes (1) a matrix with Pearson product-moment correlation for continuous variables, multiple correlation coefficient for categorical and continuous variables, and Phi coefficient and Cramer's *V* for categorical variables to identify variables related to the incomplete variable (i.e., correlates of incomplete variables), (2) a matrix with Cohen's d, Phi coefficient and Cramer's *V* for comparing cases with and without missing values, and (3) semi-partial correlations of an outcome variable conditional on the predictor variables of a substantive model with a set of candidate auxiliary variables to identify correlates of an incomplete outcome variable as suggested by Raykov and West (2016).

Usage

```
na.auxiliary(data, ..., model = NULL, categ = NULL, estimator = c("ML", "MLR"),
          missing = c("fiml", "two.stage", "robust.two.stage", "doubly.robust"),
             adjust = TRUE, weighted = FALSE, correct = FALSE,
             tri = c("both", "lower", "upper"), digits = 2, p.digits = 3,
             as.na = NULL, write = NULL, append = TRUE,
             check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with incomplete data, where missing values are coded as `NA`.
`...`	an expression indicating the variable names in `data`, e.g., `na.auxiliary(dat, x1, x2, x3)`. Categorical variables specified in the argument `categ` can be, but do not need to be selected using the `...` argument. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`model`	a character string specifying the substantive model predicting a continuous outcome variable using a set of predictor variables to estimate semi-partial correlations between the outcome variable and a set of candidate auxiliary variables. The default setting is `model = NULL`, i.e., the function computes Pearson product-moment correlation matrix and Cohen's d matrix.
`categ`	a character vector specifying the variables that are treated as categorical (see 'Details'). Note that variables that are factors or character vectors will be automatically added to the argument `categ`. Categorical variables will be excluded from the analysis when specifying the `model` argument to compute semi-partial correlations.
`estimator`	a character string indicating the estimator to be used when estimating semi-partial correlation coefficients, i.e., `"ML"` for maximum likelihood parameter estimates with conventional standard errors or `"MLR"` (default) maximum likelihood parameter estimates with Huber-White robust standard errors.
`missing`	a character string indicating how to deal with missing data when estimating semi-partial correlation coefficients, i.e., `"fiml"` for full information maximum likelihood method, `two.stage` for two-stage maximum likelihood method, `robust.two.stage` for robust two-stage maximum likelihood method, and `doubly-robust` for doubly-robust method (see 'Details' in the `item.cfa` function). The default setting is `missing = "fiml"`.

`adjust`	logical: if `TRUE` (default), phi coefficient is adjusted by relating the coefficient to the possible maximum and Cramer's *V* is corrected for small-sample bias.
`weighted`	logical: if `TRUE` (default), the weighted pooled standard deviation is used when computing Cohen's d.
`correct`	logical: if `TRUE`, correction factor for Cohen's d to remove positive bias in small samples is used.
`tri`	a character string indicating which triangular of the correlation matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	integer value indicating the number of decimal places digits to be used for displaying correlation coefficients and Cohen's d estimates.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

The function computes matrices with statistical measures depending on the level of measurement of the variables involved in the analysis:

Variables Related to the Incomplete Variable *Continuous variables*: Product-moment correlation coefficient is computed for continuous variables.

- *Continuous and categorical variable*: Multiple correlation coefficient (R) is computed based on a linear model with a dummy-coded categorical variable as predictor, where the multiple correlation coefficient is the square root of the coefficient of determination of this model. Note that the multiple R for a binary predictor variable is equivalent to the point-biserial correlation coefficient between the binary variable and the continuous outcome.
- *Categorical variables*: Phi coefficient is computed for two dichotomous variables, while Cramer's *V* is computed when one of the categorical variables is polyotomous

Variables Related to the Probability of Missingness

- *Continuous variable*: Cohen's d is computed to investigate mean differences in the continuous variable depending on cases with and without missing values.
- *Categorical variable*: Phi coefficient is computed to investigate the association between the grouping variable (0 = observed, 1 = missing) and a dichotomous variable, while Cramer's *V* is computed when the categorical variable is polytomous.

Substantive model predicting a continuous outcome variable Categorical variables are removed before computing semi-partial correlations based on the approach suggested by Raykov and West (2016).

Note that factors and characters are treated as categorical variables regardless of the specification of the argument `categ`, while numeric vectors in the data frame are treated as continuous variables if they are not specified in the argument `categ`.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`model`	lavaan model syntax for estimating the semi-partial correlations
`model.fit`	fitted lavaan model for estimating the semi-partial correlations
`args`	pecification of function arguments
`result`	list with result tables

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

Raykov, T., & West, B. T. (2016). On enhancing plausibility of the missing at random assumption in incomplete data analyses via evaluation of response-auxiliary variable correlations. *Structural Equation Modeling, 23*(1), 45–53. https://doi.org/10.1080/10705511.2014.937848

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`as.na`, `na.as`, `na.coverage`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`, `na.test`

Examples

```
# Example 1a: Auxiliary variables
na.auxiliary(airquality)

# Example 1b: Auxiliary variables, "Month" as categorical variable
na.auxiliary(airquality, categ = "Month")

# Example 2: Semi-partial correlation coefficients
na.auxiliary(airquality, model = "Ozone ~ Solar.R + Wind")
```

```
## Not run:
# Example 3a: Write Results into a text file
na.auxiliary(airquality, write = "NA_Auxiliary.txt")

# Example 3a: Write Results into an Excel file
na.auxiliary(airquality, write = "NA_Auxiliary.xlsx")

## End(Not run)
```

`na.coverage` *Variance-Covariance Coverage*

Description

This function computes the proportion of cases that contributes for the calculation of each variance and covariance.

Usage

```
na.coverage(data, ..., tri = c("both", "lower", "upper"), digits = 2,
            as.na = NULL, write = NULL, append = TRUE, check = TRUE,
            output = TRUE)
```

Arguments

`data`	a data frame with incomplete data, where missing values are coded as `NA`.
`...`	an expression indicating the variable names in `data`, e.g., `na.coverage(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`tri`	a character vector indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places to be used for displaying proportions.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`write`	a character string naming a file for writing the output into either a text file with file extension `".txt"` (e.g., `"Output.txt"`) or Excel file with file extension `".xlsx"` (e.g., `"Output.xlsx"`). If the file name does not contain any file extension, an Excel file will be written.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`result`	result table

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`write.result`, `as.na`, `na.as`, `na.auxiliary`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`, `na.test`

Examples

```
# Example 1: Compute variance-covariance coverage
na.coverage(airquality)

## Not run:
# Example 2a: Write Results into a text file
na.coverage(airquality, write = "Coverage.txt")

# Example 2b: Write Results into a Excel file
na.coverage(airquality, write = "Coverage.xlsx")

## End(Not run)
```

na.descript *Descriptive Statistics for Missing Data in Single-Level, Two-Level and Three-Level Data*

Description

This function computes descriptive statistics for missing data in single-level, two-level, and three-level data, e.g. number of incomplete cases, number of missing values, and summary statistics for the number of missing values across all variables.

Usage

```
na.descript(data, ..., cluster = NULL,  table = FALSE, digits = 2,
            as.na = NULL, write = NULL, append = TRUE, check = TRUE,
            output = TRUE)
```

Arguments

data	a data frame with incomplete data, where missing values are coded as NA.
...	an expression indicating the variable names in data, e.g., na.descript(dat, x1, x2, x3). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
cluster	a character string indicating the name of the cluster variable in data for two-level data, a character vector indicating the names of the cluster variables in data for three-level data, or a vector or data frame representing the nested grouping structure (i.e., group or cluster variables). Alternatively, a character vector indicating the variable name(s) of the cluster variable(s) in data. Note that the cluster variable at Level 3 come first in a three-level model, i.e., cluster = c("level3", "level2").
table	logical: if TRUE, a frequency table with number of observed values ("nOb"), percent of observed values ("pOb"), number of missing values ("nNA"), and percent of missing values ("pNA") is printed for each variable on the console.
digits	an integer value indicating the number of decimal places to be used for displaying percentages.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	data frame used for the current analysis
`args`	specification of function arguments
`result`	list with results

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`write.result`, `as.na`, `na.as`, `na.auxiliary`, `na.coverage`, `na.indicator`, `na.pattern`, `na.prop`, `na.test`

Examples

```
#~~~
```

```
# Three-Level Data

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

# Example 5: Descriptive statistics for missing data
na.descript(Demo.threelevel, cluster = c("cluster3", "cluster2"))

## Not run:
#-----------------------------------------------------------------------------
# Write Results

# Example 6a: Write Results into a text file
na.descript(airquality, table = TRUE, write = "NA_Descriptives.txt")

# Example 6b: Write Results into a Excel file
na.descript(airquality, table = TRUE, write = "NA_Descriptives.xlsx")

## End(Not run)
```

`na.indicator` *Missing Data Indicator Matrix*

Description

This function creates a missing data indicator matrix R that denotes whether values are observed or missing, i.e., $r = 0$ if a value is observed, and $r = 1$ if a value is missing.

Usage

```
na.indicator(data, ..., na = 1, append = TRUE, name = ".i", as.na = NULL,
             check = TRUE)
```

Arguments

`data`	a data frame with incomplete data, where missing values are coded as `NA`.
`...`	an expression indicating the variable names in `data`, e.g., `na.indicator(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`na`	an integer value specifying the value representing missing values, i.e., either `na = 0` for `0 = missing` and `1 = observed`, or `na = 1` (default) for `0` (observed) and `1 = missing`.
`append`	logical: if `TRUE` (default), missing data indicator matrix is appended to the data frame specified in the argument `data`.
`name`	a character string indicating the name suffix of indicator variables By default, the indicator variables are named with the ending `".i"` resulting in e.g. `"x1.i"` and `"x2.i"`. Note that when selecting one single variable, the indicator variable is named `x.i` by default or named after the argument `name`.

as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a matrix or data frame with $r = 1$ if a value is observed, and $r = 0$ if a value is missing.

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

as.na, na.as, na.auxiliary, na.coverage, na.descript, na.pattern, na.prop, na.test

Examples

```
# Example 1: Create missing data indicator matrix
na.indicator(airquality)

# Example 2: Do not append missing data indicator matrix to the data frame
na.indicator(airquality, append = FALSE)
```

---

na.pattern *Missing Data Pattern*

Description

This function computes a summary of missing data patterns, i.e., number (cases with a specific missing data pattern and plots the missing data patterns.

Usage

```
na.pattern(data, ..., order = FALSE, n.pattern = NULL, digits = 2, as.na = NULL,
           plot = FALSE, geom = c("all", "square", "rotate"), filename = NULL,
           width = NA, height = NA,  dpi = 600, write = NULL, append = TRUE,
           check = TRUE, output = TRUE)
```

Arguments

data	a data frame with incomplete data, where missing values are coded as NA.
...	an expression indicating the variable names in data e.g., na.pattern(dat, x1, x2, x3). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
order	logical: if TRUE, variables are ordered from left to right in increasing order of missing values.
n.pattern	an integer value indicating the minimum number of cases sharing a missing data pattern to be included in the result table and the plot, e.g., specifying n.pattern = 5 excludes missing data patters with less than 5 cases.
digits	an integer value indicating the number of decimal places to be used for displaying percentages.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	logical: if TRUE, missing data pattern is plotted.
geom	a character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "square" the plot tiles are squares to mimic the md.pattern function in the package **mice**, and "rotate" for rotating variable name labels 90 degrees.
filename	a character string indicating the filename argument (default is "NA_Pattern.pdf") including the file extension for the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis

data	data frame with variables used in the analysis
args	specification of function arguments
result	result table
plot	ggplot2 object for plotting the results
pattern	a numeric vector indicating the missing data pattern for each case

Note

The code for plotting missing data patterns is based on the `plot_pattern` function in the **ggmice** package by Hanne Oberman.

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

Oberman, H. (2023). *ggmice: Visualizations for 'mice' with 'ggplot2'*. R package version 0.1.0. https://doi.org/10.32614/CRAN.package.ggmice

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`write.result`, `as.na`, `na.as`, `na.auxiliary`, `na.coverage`, `na.descript`, `na.indicator`, `na.prop`, `na.test`

Examples

```
## Not run:
#-----------------------------------------------------------------------
# Missing Data Pattern

# Example 1a: Compute a summary of missing data patterns
dat.pattern <- na.pattern(airquality)

# Example 1b: Exclude missing data patterns with less than 3 cases
na.pattern(airquality, plot = TRUE, n.pattern = 3)

# Example 1c: Vector of missing data pattern for each case
dat.pattern$pattern

# Example 1d: Data frame without cases with missing data pattern 2
airquality[dat.pattern$pattern != 2, ]

#-----------------------------------------------------------------------
# Plot
```

```
# Example 2a: Plot a summary of missing data patterns
na.pattern(airquality, plot = TRUE)

# Example 2b: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- na.pattern(airquality)
plot(object, tile.alpha = 0.8, scale.fill.col = c("red2", "darkgreen"))

#-----------------------------------------------------------------------
# Write Results

# Example 3a: Write Results into a text file
na.pattern(airquality, write = "NA_Pattern.xlsx")

# Example 3b: Write Results into a Excel file
na.pattern(airquality, write = "NA_Pattern.xlsx")

# Example 3c: Write Results into a text file
na.pattern(airquality, plot = TRUE, filename = "NA_Pattern.png", width = 8, height = 7)
## End(Not run)
```

`na.prop` *Proportion of Missing Data for Each Case*

Description

This function computes the proportion of missing data for each case in a data frame.

Usage

```
na.prop(data, ..., digits = 2, append = TRUE, name = "na.prop",
        as.na = NULL, check = TRUE)
```

Arguments

`data`	a data frame with incomplete data, where missing values are coded as `NA`.
`...`	an expression indicating the variable names in `data`, e.g., `na.prop(dat, x1, x2, x3)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`name`	a character string indicating the name of the variable appended to the data frame specified in the argument `data` when `append = TRUE`.
.	
`append`	logical: if `TRUE` (default), variable with proportion of missing data is appended to the data frame specified in the argument `data`
`digits`	an integer value indicating the number of decimal places to be used for displaying proportions.

as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.

Value

Returns a numeric vector with the same length as the number of rows in data containing the proportion of missing data.

Author(s)

Takuya Yanagida

References

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

as.na, na.as, na.auxiliary, na.coverage, na.descript, na.indicator, na.pattern, na.test

Examples

```
# Example 1: Compute proportion of missing data for each case in the data frame
na.prop(airquality)

# Example 2: Do not append proportions of missing data to the data frame
na.prop(airquality, append = FALSE)
```

---

na.satcor *Fit a Saturated Correlates Model*

Description

This function estimates a confirmatory factor analysis model (cfa.satcor function), structural equation model (sem.satcor function), growth curve model (growth.satcor function), or latent variable model (lavaan.satcor function) in the R package **lavaan** using full information maximum likelihood (FIML) method to handle missing data while automatically specifying a saturated correlates model to incorporate auxiliary variables into a substantive model without affecting the parameter estimates, the standard errors, or the estimates of quality of fit (Graham, 2003).

Usage

```
na.satcor(model, data, aux, fun = c("cfa", "sem", "growth", "lavaan"),
          check = TRUE, ...)

cfa.satcor(model, data, aux, check = TRUE, ...)

sem.satcor(model, data, aux, check = TRUE, ...)

growth.satcor(model, data, aux, check = TRUE, ...)

lavaan.satcor(model, data, aux, check = TRUE, ...)
```

Arguments

`model`	a character string indicating the lavaan model syntax without the auxiliary variables specified in `aux`.
`data`	a data frame containing the observed variables used in the lavaan model syntax specified in `model` and the auxiliary variables specified in `aux`.
`aux`	a character vector indicating the names of the auxiliary variables in the data frame specified in `data` that will be added to the lavaan model syntax specified in `model`. Note that this function can only incorporate continuous auxiliary variables, i.e., the function cannot deal with categorical auxiliary variables.
`fun`	a character string indicating the name of a specific lavaan function used to fit `model`, i.e., `cfa`, `sem`, `growth`, or `lavaan`. Note that this argument is only required for the function `na.satcor`.
`check`	logical: if `TRUE` (default), argument specification is checked.
`...`	additional arguments passed to the lavaan function.

Value

An object of class lavaan, for which several methods are available in the R package **lavaan**, including a summary method.

Note

This function is a modified copy of the `auxiliary()`, `cfa.auxiliary()`, `sem.auxiliary()`, `growth.auxiliary()`, and `lavaan.auxiliary()` functions in the **semTools** package by Terrence D. Jorgensen et al. (2022).

Author(s)

Takuya Yanagida

References

Graham, J. W. (2003). Adding missing-data-relevant variables to FIML-based structural equation models. *Structural Equation Modeling, 10*(1), 80-100. https://doi.org/10.1207/S15328007SEM1001_4

Jorgensen, T. D., Pornprasertmanit, S., Schoemann, A. M., & Rosseel, Y. (2022). *semTools: Useful tools for structural equation modeling*. R package version 0.5-6. Retrieved from https://CRAN.R-project.org/package=semTools

Examples

```
# Load lavaan package
library(lavaan)

#————————————————————————————————————————————————————————————————————————
# Example 1: Saturated correlates model for the sem function

# Model specification
model <- 'Ozone ~ Wind'

# Model estimation using the sem.satcor function
mod.fit <- sem.satcor(model, data = airquality, aux = c("Temp", "Month"))

# Model estimation using the na.satcor function
mod.fit <- na.satcor(model, data = airquality, fun = "sem", aux = c("Temp", "Month"),
                     estimator = "MLR")

# Result summary
summary(mod.fit)
```

`na.test` *Missing Completely at Random (MCAR) Test*

Description

This function performs Little's Missing Completely at Random (MCAR) test and Jamshidian and Jalal's approach for testing the MCAR assumption. By default, the function performs the Little's MCAR test.

Usage

```
na.test(data, ..., print = c("all", "little", "jamjal"),
        impdat = NULL, delete = 6, method = c("npar", "normal"),
        m = 20, seed = 42, nrep = 10000, n.min = 30,
        pool = c("m", "med", "min", "max", "random"),
        alpha = 0.05, digits = 2, p.digits = 3, as.na = NULL,
        write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

`data`	a data frame with incomplete data, where missing values are coded as `NA`.
`...`	an expression indicating the variable names in `data`, e.g., `na.test(dat, x1, x2, x3)`. Note that the operators `.`, `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.

`print` a character vector indicating which results to be printed on the console, i.e. `"all"` for Little's MCAR test and Jamshidian and Jalal's approach, `"little"` (default) for Little's MCAR test, and `"jamjal"` for Jamshidian and Jalal's approach.

`impdat` an object of class `mids` from the **mice** package to provide a data set multiply imputed in the **mice** package. The function will not impute the data data set specified in the argument `data` when specifying this argument and will use the imputed data sets provided in the argument `impdat` for performing the Jamshidian and Jalal's approach. Note that the argument `data` still needs to be specified because the variables used for the analysis are extracted from the data frame specified in `data`.

`delete` an integer value indicating missing data patterns consisting of `delete` number of cases or less removed from the Jamshidian and Jalal's approach. The default setting is `delete = 6`.

`method` a character string indicating the imputation method, i.e., `"npar"` for using a non-parametric imputation method by Sirvastava and Dolatabadi (2009) or `"normal"` for imputing missing data assuming that the data come from a multivariate normal distribution (see Jamshidian & Jalal, 2010).

`m` an integer value indicating the number of multiple imputations. The default setting is `m = 20`.

`seed` an integer value that is used as argument by the `set.seed` function for offsetting the random number generator before performing Jamshidian and Jalal's approach. The default setting is `seed = 42`. Set the value to `NULL` to specify a system selected seed.

`nrep` an integer value indicating the replications used to simulate the Neyman distribution to determine the cut off value for the Neyman test. Larger values increase the accuracy of the Neyman test. The default setting is `nrep = 10000`.

`n.min` an integer value indicating the minimum number of cases in a group that triggers the use of asymptotic Chi-square distribution in place of the empirical distribution in the Neyman test of uniformity.

`pool` a character string indicating the pooling method, i.e., `"m"` for computing the average test statistic and p-values, `"med"` for computing the median test statistic and p-values, `"min"` for computing the maximum test statistic and minimum p-values, `"max"` for computing the minimum test statistic and maximum p-values, and `"random"` for randomly choosing a test statistic and corresponding p-value from repeated complete data analyses. The default setting is `pool = "med"`, i.e., median test statistic and p-values are computed as suggested by Eekhout, Wiel and Heymans (2017).

`alpha` a numeric value between 0 and 1 indicating the significance level of the Hawkins test. The default setting is `alpha = 0.05`, i.e., the Anderson-Darling non-parametric test is provided when the p-value of the Hawkins test is less than or equal `0.05`.

`digits` an integer value indicating the number of decimal places to be used for displaying results.

`p.digits` an integer value indicating the number of decimal places to be used for displaying the *p*-value.

`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
`write`	a character string naming a text file with file extension `".txt"` (e.g., `"Output.txt"`) for writing the output into a text file.
`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown.

Details

Little's MCAR Test Little (1988) proposed a multivariate test of Missing Completely at Random (MCAR) that tests for mean differences on every variable in the data set across subgroups that share the same missing data pattern by comparing the observed variable means for each pattern of missing data with the expected population means estimated using the expectation-maximization (EM) algorithm (i.e., EM maximum likelihood estimates). The test statistic is the sum of the squared standardized differences between the subsample means and the expected population means weighted by the estimated variance-covariance matrix and the number of observations within each subgroup (Enders, 2010). Under the null hypothesis that data are MCAR, the test statistic follows asymptotically a chi-square distribution with $\sum k_j - k$ degrees of freedom, where k_j is the number of complete variables for missing data pattern j, and k is the total number of variables. A statistically significant result provides evidence against MCAR.

Note that Little's MCAR test has a number of problems (see Enders, 2010).

- **First**, the test does not identify the specific variables that violates MCAR, i.e., the test does not identify potential correlates of missingness (i.e., auxiliary variables).
- **Second**, the test is based on multivariate normality, i.e., under departure from the normality assumption the test might be unreliable unless the sample size is large and is not suitable for categorical variables.
- **Third**, the test investigates mean differences assuming that the missing data pattern share a common covariance matrix, i.e., the test cannot detect covariance-based deviations from MCAR stemming from a Missing at Random (MAR) or Missing Not at Random (MNAR) mechanism because MAR and MNAR mechanisms can also produce missing data subgroups with equal means.
- **Fourth**, simulation studies suggest that Little's MCAR test suffers from low statistical power, particularly when the number of variables that violate MCAR is small, the relationship between the data and missingness is weak, or the data are MNAR (Thoemmes & Enders, 2007).
- **Fifth**, the test can only reject, but cannot prove the MCAR assumption, i.e., a statistically not significant result and failing to reject the null hypothesis of the MCAR test does not prove the null hypothesis that the data is MCAR.
- **Sixth**, under the null hypothesis the data are actually MCAR or MNAR, while a statistically significant result indicates that missing data are MAR or MNAR, i.e., MNAR cannot be ruled out regardless of the result of the test.

The function for performing Little's MCAR test is based on the `mlest` function from the **mvnmle** package which can handle up to 50 variables. Note that the `mcar_test` function in

the **naniar** package is based on the `prelim.norm` function from the **norm** package. This function can handle about 30 variables, but with more than 30 variables specified in the argument `data`, the `prelim.norm` function might run into numerical problems leading to results that are not trustworthy (i.e., `p.value = 1`). In that case, the warning message `In norm::prelim.norm(data) : NAs introduced by coercion to integer range` is printed on the console.

Jamshidian and Jalal's Approach for Testing MCAR Jamshidian and Jalal (2010) proposed an approach for testing the Missing Completely at Random (MCAR) assumption based on two tests of multivariate normality and homogeneity of covariances among groups of cases with identical missing data patterns:

1. **In the first step**, missing data are multiply imputed (`m = 20` times by default) using a non-parametric imputation method (`method = "npar"` by default) by Sirvastava and Dolatabadi (2009) or using a parametric imputation method assuming multivariate normality of data (`method = "normal"`) for each group of cases sharing a common missing data pattern.
2. **In the second step**, a modified Hawkins test for multivariate normality and homogeneity of covariances applicable to complete data consisting of groups with a small number of cases is performed. A statistically not significant result indicates no evidence against multivariate normality of data or homogeneity of covariances, while a statistically significant result provides evidence against multivariate normality of data or homogeneity of covariances (i.e., violation of the MCAR assumption). Note that the Hawkins test is a test of multivariate normality as well as homogeneity of covariance. Hence, a statistically significant test is ambiguous unless the researcher assumes multivariate normality of data.
3. **In the third step**, if the Hawkins test is statistically significant, the Anderson-Darling non-parametric test is performed. A statistically not significant result indicates evidence against multivariate normality of data but no evidence against homogeneity of covariances, while a statistically significant result provides evidence against homogeneity of covariances (i.e., violation of the MCAR assumption). However, no conclusions can be made about the multivariate normality of data when the Anderson-Darling non-parametric test is statistically significant.

In summary, a statistically significant result of both the Hawkins and the Anderson-Darling non-parametric test provides evidence against the MCAR assumption. The test statistic and the significance values of the Hawkins test and the Anderson-Darling non-parametric based on multiply imputed data sets are pooled by computing the median test statistic and significance value (`pool = "med"` by default) as suggested by Eekhout, Wiel, and Heymans (2017).

Note that out of the problems listed for the Little's MCAR test the first, second (i.e., approach is not suitable for categorical variables), fifth, and sixth problems also apply to the Jamshidian and Jalal's approach for testing the MCAR assumption.

In practice, rejecting or not rejecting the MCAR assumption may not be relevant as modern missing data handling methods like full information maximum likelihood (FIML) estimation, Bayesian estimation, or multiple imputation are asymptotically valid under the missing at random (MAR) assumption (Jamshidian & Yuan, 2014). It is more important to distinguish MAR from missing not at random (MNAR), but MAR and MNAR mechanisms cannot be distinguished without auxiliary information.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	matrix or data frame specified in `data`
`args`	specification of function arguments
`result`	list with result tables, i.e., `little` for the result table of the Little's MCAR test, `jamjal` for the list with results of the Jamshidian and Jalal's approach, `hawkins` for the result table of the Hawkins test, and `anderson` for the result table of the Anderson-Darling non-parametric test

Note

The code for Little's MCAR test is a modified copy of the `LittleMCAR` function in the **BaylorEdPsych** package by A. Alexander Beaujean. The code for Jamshidian and Jalal's approach is a modified copy of the `TestMCARNormality` function in the **MissMech** package by Mortaza Jamshidian, Siavash Jalal, Camden Jansen, and Mao Kobayashi (2024).

Author(s)

Takuya Yanagida

References

Beaujean, A. A. (2012). *BaylorEdPsych: R Package for Baylor University Educational Psychology Quantitative Courses*. R package version 0.5. http://cran.nexr.com/web/packages/BaylorEdPsych/index.html

Eekhout, I., M. A. Wiel, & M. W. Heymans (2017). Methods for significance testing of categorical covariates in logistic regression models after multiple imputation: Power and applicability analysis. *BMC Medical Research Methodology*, 17:129. https://doi.org/10.1186/s12874-017-0404-7

Enders, C. K. (2022). *Applied missing data analysis* (2nd ed.). The Guilford Press.

Little, R. J. A. (1988). A test of Missing Completely at Random for multivariate data with missing values. *Journal of the American Statistical Association, 83*, 1198-1202. https://doi.org/10.2307/2290157

Jamshidian, M., & Jalal, S. (2010). Tests of homoscedasticity, normality, and missing completely at random for incomplete multivariate data. *Psychometrika, 75*(4), 649-674. https://doi.org/10.1007/s11336-010-9175-3

Jamshidian, M., & Yuan, K.H. (2014). Examining missing data mechanisms via homogeneity of parameters, homogeneity of distributions, and multivariate normality. *WIREs Computational Statistics, 6*(1), 56-73. https://doi.org/10.1002/wics.1287

Mortaza, J., Siavash, J., Camden, J., & Kobayashi, M. (2024). *MissMech: Testing Homoscedasticity, Multivariate Normality, and Missing Completely at Random*. R package version 1.0.4. https://doi.org/10.32614/CRAN.package.MissMech

Srivastava, M.S., & Dolatabadi, M. (2009). Multiple imputation and other resampling scheme for imputing missing observations. *Journal of Multivariate Analysis, 100*, 1919-1937. https://doi.org/10.1016/j.jmva.2009.06.003

Thoemmes, F., & Enders, C. K. (2007, April). *A structural equation model for testing whether data are missing completely at random*. Paper presented at the annual meeting of the American Educational Research Association, Chicago, IL.

See Also

as.na, na.as, na.auxiliary, na.coverage, na.descript, na.indicator, na.pattern, na.prop.

Examples

```
# Example 1: Perform Little's MCAR test
na.test(airquality)

# Example 2: Perform Jamshidian and Jalal's approach
na.test(airquality, print = "jamjal")

## Not run:
# Example 3: Write results into a text file
na.test(airquality, write = "NA_Test.txt")
## End(Not run)
```

---

plot.misty.object *Plots misty.object object*

Description

This function plots an `misty.object` object.

Usage

```
## S3 method for class 'misty.object'
plot(x, plot = NULL, geom = NULL, color = NULL, type = NULL,
     line.type = NULL, line.width = NULL, line.col = NULL,
     ci.col = NULL, ci.width = NULL, ci.type = NULL,
     errorbar.width = NULL, errorbar.dodge.width = NULL,
     smooth.fill = NULL, smooth.alpha = NULL, smooth.type = NULL,
     smooth.width = NULL, smooth.col = NULL,
     point.size = NULL, point.width = NULL, point.type = NULL,
     point.shape = NULL, point.col = NULL, point.fill = NULL,
     jitter.size = NULL, jitter.width = NULL,
     jitter.height = NULL, jitter.alpha = NULL,
     ribbon.fill = NULL, ribbon.col = NULL, ribbon.alpha = NULL,
     violin.alpha = NULL, violin.trim = NULL,
     density.col = NULL, density.width = NULL,
     density.type = NULL, density.alpha = NULL,
     hist.binwidth = NULL, hist.bins = NULL, hist.col = NULL,
     hist.fill = NULL, hist.alpha = NULL,
     box.width = NULL, box.alpha = NULL, tile.col = NULL, tile.alpha = NULL,
     function.type = NULL, function.width = NULL, function.col = NULL,
     abline.col = NULL, abline.width = NULL,
    hline.intercept = NULL, hline.type = NULL, hline.width = NULL, hline.col = NULL,
    vline.intercept = NULL, vline.type = NULL, vline.width = NULL, vline.col = NULL,
```

```
text.size = NULL, text.col = NULL, text.font = NULL,
gray = NULL, start = NULL, end = NULL,
xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
xbreaks = NULL, ybreaks = NULL,
axis.title.size = NULL, axis.text.size = NULL, strip.text.size = NULL,
scale.fill.col = NULL, scale.fill.low = NULL, scale.fill.high = NULL,
scale.fill.mid = NULL, title = NULL, subtitle = NULL, group.col = NULL,
plot.margin = NULL, legend.title = NULL, legend.position = NULL,
legend.box.margin = NULL, legend.key.size = NULL, legend.text.size = NULL,
facet.ncol = NULL, facet.nrow = NULL, facet.scales = NULL,
conf.level = NULL, adjust = NULL,
filename = NULL, width = NULL, height = NULL, units = NULL, dpi = NULL,
check = TRUE, ...)
```

Arguments

x	`misty.object` object.
plot	see 'Details'
geom	see 'Details'
color	see 'Details'
type	see 'Details'
line.type	see 'Details'
line.width	see 'Details'
line.col	see 'Details'
ci.col	see 'Details'
ci.width	see 'Details'
ci.type	see 'Details'
errorbar.width	see 'Details'
errorbar.dodge.width	see 'Details'
smooth.fill	see 'Details'
smooth.alpha	see 'Details'
smooth.type	see 'Details'
smooth.width	see 'Details'
smooth.col	see 'Details'
point.size	see 'Details'
point.width	see 'Details'
point.type	see 'Details'
point.shape	see 'Details'
point.col	see 'Details'
point.fill	see 'Details'
jitter.size	see 'Details'

jitter.width	see 'Details'
jitter.height	see 'Details'
jitter.alpha	see 'Details'
ribbon.fill	see 'Details'
ribbon.col	see 'Details'
ribbon.alpha	see 'Details'
violin.alpha	see 'Details'
violin.trim	see 'Details'
density.col	see 'Details'
density.width	see 'Details'
density.type	see 'Details'
density.alpha	see 'Details'
hist.binwidth	see 'Details'
hist.bins	see 'Details'
hist.col	see 'Details'
hist.fill	see 'Details'
hist.alpha	see 'Details'
box.width	see 'Details'
box.alpha	see 'Details'
tile.col	see 'Details'
tile.alpha	see 'Details'
function.type	see 'Details'
function.width	see 'Details'
function.col	see 'Details'
abline.col	see 'Details'
abline.width	see 'Details'
hline.intercept	see 'Details'
hline.type	see 'Details'
hline.width	see 'Details'
hline.col	see 'Details'
vline.intercept	see 'Details'
vline.type	see 'Details'
vline.width	see 'Details'
vline.col	see 'Details'
text.size	see 'Details'
text.col	see 'Details'

text.font	see 'Details'
gray	see 'Details'
start	see 'Details'
end	see 'Details'
xlab	see 'Details'
ylab	see 'Details'
xlim	see 'Details'
ylim	see 'Details'
xbreaks	see 'Details'
ybreaks	see 'Details'
axis.title.size	see 'Details'
axis.text.size	see 'Details'
strip.text.size	see 'Details'
scale.fill.col	see 'Details'
scale.fill.low	see 'Details'
scale.fill.high	see 'Details'
scale.fill.mid	see 'Details'
title	see 'Details'
subtitle	see 'Details'
group.col	see 'Details'
plot.margin	see 'Details'
legend.title	see 'Details'
legend.position	see 'Details'
legend.box.margin	see 'Details'
legend.key.size	see 'Details'
legend.text.size	see 'Details'
facet.ncol	see 'Details'
facet.nrow	see 'Details'
facet.scales	see 'Details'
conf.level	see 'Details'
adjust	see 'Details'
filename	a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"` (default), `".jpeg"`, `".tiff"`, `".png"`, `".bmp"`, `".svg"` or `".wmf"` needs to be specified as file extension in the `file` argument.

width	a numeric value indicating the width argument for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
units	a character string indicating the units argument in the ggsave function. Note that one of "in", "cm", "mm", or "px" needs to be specified.
dpi	a numeric value indicating the dpi argument for the ggsave function.
check	logical: if TRUE (default), argument specification is checked.
...	further arguments passed to or from other methods.

Details

This function provides plotting arguments depending on the type of the output object specified for the argument x. Note that the arguments filename, width, height, units, and dpi are available for all types of the output object.

Functions aov.b **and** aov.w The plot function for the misty object of type "aov.b", "aov.w", and test.welch has following plotting arguments:

- geom: a character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "bar" for drawing bars representing means for each groups, "point" for drawing points representing means for each group, "line" for drawing a line connecting means of each group and lines connecting data points when requesting "jitter", "ci" for drawing error bars representing confidence intervals, and "jitter" for drawing jittered data points.
- conf.level: a numeric value between 0 and 1 (default: 0.95) indicating the confidence level of the interval.
- adjust: logical: if TRUE (default), difference-adjustment for the confidence intervals is applied.
- point.size: a numeric value (default: 3) indicating the size argument in the geom_point function.
- line.type: a numeric value (default: "solid") indicating the linetype argument for the geom_line function. Note that this argument is only available for the "aov.w" function.
- line.width: a numeric value (default: 0.5) indicating the linewidth argument for the geom_line function. Note that this argument is only available for the "aov.w" function.
- errorbar.width: a numeric value (default: 0.1) indicating the width argument for the geom_errorbar function.
- jitter.size: a numeric value (default: 1.25) indicating the size argument for the geom_jitter function.
- jitter.width: a numeric value (default: 0.05) indicating the width argument for the geom_jitter function.
- jitter.height: a numeric value (default: 0) indicating the height argument for the geom_jitter function
- jitter.alpha: a numeric value between 0 and 1 (default: 0.1) the alpha argument for the geom_jitter function
- xlab: a character string (default: NULL) indicating the name argument in the scale_x_discrete function.

- ylab: a character string (default: "y") indicating the name argument in the scale_y_continuous function.
- ylim: a numeric vector with two elements (default: NULL) indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.
- title: a character string (default: "") indicating the title argument in the labs function.
- subtitle: a character string indicating the subtitle argument in the labs function. The default is "Two-Sided Confidence Interval" when adjust = FALSE or "Two-Sided Difference-Adjusted Confidence Interval" for the aov.b function and "Two-Sided Difference-Adjusted Cousineau-Morey Confidence Interval Confidence Interval" for the aov.w function when adjust = TRUE.

Function boot.bs The ggplot2 arguments provided for the function boot.bs are described in the help page of the boot.bs function.

Function check.outlier The plot function for the misty object of type "check.outlier" has following plotting arguments:

- type: character vector indicating which statistical measures to include in the plot, i.e., "mahal" for the Mahalanobis distance, "hat" for the hat value, "rstand" for the standardized leverage-corrected residual, "rstud" for the studentized leverage-corrected residual, "cook" for the Cook's distance, and "dfb" for the DFBETAS.
- geom: a character vector indicating which plot elements to draw, i.e., "none" for not drawing any labels, "text" (default) for drawing labels above or below each observation.
- point.size: a numeric value (default: 2) indicating the size argument in the geom_point function.
- point.shape: a numeric value (default: 21) indicating the shape argument in the geom_point function.
- point.fill: a character string (default: "#648FFF") indicating the fill argument in the geom_point function.
- point.col: a character string (default: "black") indicating the color argument in the geom_point function.
- line.type: an integer value or character string (default: "solid") indicating the linetype argument in the geom_linerange function.
- line.width: a numeric value (default: 0.5) indicating the linewidth argument in the geom_linerange function.
- line.col: a character string (default: "#648FFF") indicating the color argument in the geom_linerange function.
- hline.type: an integer value or character string (default: "dashed") indicating the linetype argument in the geom_hline function.
- hline.width: a numeric value (default: 0.5) indicating the linewidth argument in the geom_hline function.
- hline.col: a character string (default: "#FE6100") indicating the color argument in the geom_hline function. the color argument in the geom_hline function.
- text.size: a numeric value (default: 3) indicating the size argument in the geom_text function.

- text.col: a character string (default: "black") indicating the color argument in the geom_text function.
- ylim: a numeric vector with two elements indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.
- axis.title.size: a numeric value (default: 11) indicating the size argument in the element_text function, i.e. theme(axis.title = element_text(size = axis.text.size))
- axis.text.size: a numeric value (default: 10) indicating the size argument in the element_text function, i.e., theme(axis.text = element_text(size = axis.text.size))

Function check.resid The plot function for the misty object of type "check.resid" have the same arguments as the test.linear function when type = "linear", the test.homo function when type = "homo", and the test.normal function when type = "test.norm". In addition, the plot function for the misty object of type "check.resid" has following plotting argument:

- geom: a character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "lm" for drawing the regression line, "loess" for drawing the LOESS smoothing curve, "ci.lm" for drawing confidence bands around the regression line, "ci.loess" for drawing confidence bands around the LOESS smoothing curve, "hline" or drawing a horizontal line at 0, "qq" for plotting Q-Q plots, and "hist" for plotting histograms. Note that the arguments "lm", "loess", "ci.lm", and "ci.less" are applicable when type = "linear", the arguments "hline", "loess", and "ci.loess" are applicable when type = "homo", and the arguments "lqq" and "hist" are applicable when type = "norm".

Functions ci.* The plot function for the misty object of type "ci.cor", "ci.mean", "ci.median", "ci.prop", "ci.var", and "ci.sd" has following plotting arguments:

- plot: a character string indicating the type of the plot to display, i.e., "ci" (default) for displaying confidence intervals or "boot" for displaying bootstrap samples with histograms and density curves when the argument.
- geom: a character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements,"hist" for drawing histograms when plotting bootstrap samples (plot = "boot"), "density" for drawing density curves when plotting bootstrap samples (plot = "boot"), "point" for drawing vertical lines representing the point estimate of the correlation coefficients when plotting bootstrap samples (plot = "boot"), "ci" for drawing vertical lines representing the bootstrap confidence intervals of the correlation when plotting bootstrap samples (plot = "boot"), and "line" for drawing a horizontal line when specifying plot = "ci" or a vertical line when specifying plot = "boot".
- point.size: a numeric value (default: 2.5) indicating the size argument in the geom_point function.
- point.shape: a numeric value between 0 and 25 (default: 19) indicating the shape argument in the geom_point function.
- errorbar.width: a numeric value (default: 0.3) indicating the width argument in the geom_errorbar function.
- errorbar.dodge.width: a numeric value (default: 0.5) indicating the width argument in the geom_errorbar function.

- hist.bins: a numeric value or a function (default: NULL) indicating the bins argument in the geom_histogram function.
- hist.binwidth: a numeric value or a function (default: NULL) indicating the binwidth argument in the geom_histogram function.
- hist.alpha: a numeric value between 0 and 1 (default: 0.4) indicating the alpha argument in the geom_histogram function.
- hist.fill: a character string (default: "gray85") indicating the fill argument in the geom_histogram function.
- density.col: a character string (default: "#0072B2") indicating the color argument in the geom_density function.
- density.width: a numeric value (default: 0.5) indicating the linewidth argument in the geom_density function.
- density.type: a numeric value or character string (default: 0.5) indicating the linetype argument in the geom_density function.
- point.col: a character string (default: "#CC79A7") indicating the color argument in the geom_vline function. Note that this argument applied only when no grouping variable was specified group = NULL.
- point.width: a numeric value (default: 0.6) indicating the linewdith argument in the geom_vline function.
- point.type: a numeric value or character string (default: "solid") indicating the linetype argument in the geom_vline function.
- ci.col: character string (default: "black") indicating the color argument in the geom_vline function. Note that this argument applied only when no grouping variable was specified group = NULL.
- ci.width: a numeric value (default: 0.6) indicating the linewdith argument in the geom_vline function.
- ci.type: a numeric value or character string (default: "dashed") indicating the linetype argument in the geom_vline function.
- hline.intercept: a numeric value (default = 0) indicating the yintercept argument in the geom_hline function.
- hline.type: a character string (default: "solid") indicating the linetype argument in the geom_hline function.
- hline.col: a character string (default: "gray65") indicating the color argument in the geom_hline function.
- vline.intercept: a numeric value (default = 0) indicating the xintercept argument in the geom_vline function.
- vline.type: a character string (default: "solid") indicating the linetype argument in the geom_vline function.
- vline.col: a character string (default: "gray65") indicating the color argument in the geom_vline function.
- xlab: a character string indicating the name argument in the scale_x_continuous function. The default setting is xlab = NULL when plot = "ci" and xlab = "Correlation Coefficient", xlab = "Arithmetic Mean", xlab = "Median", xlab = "Proportion", xlab = "Variance", or xlab = "Standard Deviation".
- ylab: a character string indicating the name argument in the scale_y_continuous function. The default setting is ylab = "Correlation Coefficient", ylab = "Arithmetic

Mean", ylab = "Median", ylab = "Proportion", ylab = "Variance", or ylab = "Standard Deviation" when plot = "ci" and ylab = "Probability Density f(x)" when plot = "boot".

- xlim: a numeric vector with two elements indicating the limits argument in the scale_x_continuous function. The default setting is xlim = NULL when plot = "ci" and xlim = c(-1, 1) for the correlation coefficient and proportion or xlim = NULL) for the arithmetic mean, median, variance and standard deviation when plot = "boot".
- ylim: a numeric vector with two elements indicating the limits argument in the scale_y_continuous function. The default setting is ylim = c(-1, 1) for the correlation coefficient and proportion and ylim = NULL for the arithmetic mean, median, variance and standard deviation when plot = "ci" and xlim = NULL when plot = "boot".
- xbreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_x_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.
- axis.title.size: a numeric value (default: 11) indicating the size argument in the element_text function, i.e. theme(axis.title = element_text(size = axis.text.size))
- axis.text.size: a numeric value (default: 10) indicating the size argument in the element_text function, i.e. theme(axis.text = element_text(size = axis.text.size)).
- strip.text.size: a numeric value (default: 11) indicating the size argument in the element_text function, i.e. theme(strip.text = element_text(size = strip.text.size)).
- title: a character string (default: NULL) indicating the title argument in the labs function.
- subtitle: a character string (default: NULL) indicating the subtite argument in the labs function.
- group.col: a character vector (default: NULL) indicating the color argument in the scale_color_manual and scale_fill_manual functions when specifying a grouping variable using the argument group.
- plot.margin: a numeric vector (default: NA) with four elements indicating the plot.margin argument in the theme function. The default setting is c(5.5, 5.5, 5.5, 5.5) but switches to c(5.5, 5.5, -2.5, 5.5) when specifying a grouping variable using the argument group.
- legend.title: a character string (default: "") indicating the color argument in the labs function when specifying a grouping variable using the argument group.
- legend.position: a character string (default: "bottom") indicating the legend.position in the theme argument when specifying a grouping variable using the argument group.
- legend.box.margin: a numeric vector (default: c(-10, 0, 0, 0)) with four elements indicating the legend.box.margin argument in the theme function when specifying a grouping variable using the argument group.
- facet.ncol: a numeric value (default: NULL) indicating the ncol argument in the facet_wrap when specifying a split variable using the argument split.
- facet.nrow: a numeric value (default: NULL) indicating the nrow argument in the facet_wrap function when specifying a split variable using the argument split.
- facet.scales: a character string (default: "free_y") indicating the scales argument in the facet_wrap function, i.e. "fixed", "free_x", "free_y" (default) or "free" when specifying a split variable using the argument split.

Functions `cor.matrix` **and** `multilevel.cor` The `plot` functions for the misty object of type `"cor.matrix"` and `"multilevel.cor"` function have following plotting arguments:

- `tile.col`: a character string (default: `"black"`) indicating the `color` argument in the `geom_tile` function.
- `scale.fill.low`: a character string (default: `"#D41159"`) indicating the `low` argument in the `scale_fill_gradient2` function.
- `scale.fill.high`: a character string (default: `"#1A85FF"`) indicating the `high` argument in the `scale_fill_gradient2` function.
- `scale.fill.mid`: a character string (default: `"#1A85FF"`) indicating the `mid` argument in the `scale_fill_gradient2` function.
- `text.size`: a numeric vector with two elements (default: `c(3.5, 3.5)`) indicating the `size` argument in the `geom_text` function.
- `text.col`: a character vector with two elements (default: `c("black", "black")`) indicating the `color` argument in the `geom_text` function.
- `text.font`: a character vector with two elements (default: `c("plain", "plain")`) indicating the `fontface` argument in the `geom_text` function.

Function `item.dfi` The ggplot2 arguments provided for the function `item.dfi` are described in the help page of the `item.dfi` function.

Functions `multilevel.r2` **and** `multilevel.r2.manual` The `plot` functions for the misty object of type `"multilevel.r2"` and `"multilevel.r2.manual"` have following plotting arguments:

- `scale.fill.col`: a character vector, graphical parameter indicating the color of bars in the bar chart in the following order: Fixed slopes (Within), Fixed slopes (Between), Slope variation (Within), Intercept variation (Between), and Residual (Within). The default setting is `c("#D55E00", "#0072B2", "#CC79A7", "#009E73", "#E69F00")`.
- `gray`: logical: if `TRUE`, graphical parameter to draw the bar chart in gray scale.
- `start`: a numeric value between 0 and 1 (default: `0.15`), graphical parameter to specify the gray value at the low end of the palette.
- `end`: a numeric value between 0 and 1 (default: `0.85`), graphical parameter to specify the gray value at the high end of the palette.

Function `na.pattern` The `plot` functions for the misty object of type `"na.pattern"` function have following plotting arguments:

- `geom`: a character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"square"` (default) for drawing squares that mimic the `md.pattern` function in the package **mice**, and `"rotate"`, to rotate variable name labels 90 degrees.
- `tile.alpha`: a numeric value between 0 and 1 (default: `0.6`) for the `alpha` argument (default is `0.1`).
- `scale.fill.col`: a character string indicating the color for the `"fill"` argument. Note that the first color represents missing values and the second color represent observed values. The default setting is `c("#B61A51B3", "#006CC2B3")`.
- `plot.margin`: a numeric vector indicating the `plot.margin` argument for the `margin` function for the `plot.margin` argument within the `theme` function. The default setting is `c(4, 16, 0, 4)`.
- `legend.box.margin`: a numeric vector indicating the `legend.box.margin` argument for the `element_text` function for the `legend.text` argument within the `theme` function. The default setting is `c(-8, 6, 6, 6)`.

- legend.key.size: a numeric value (default: unit(12, "pt")) indicating the legend.key argument for the theme function.

Function test.homo The plot function for the misty object of type "test.homo" has following plotting arguments:

- geom: a character vector indicating which plot elements to draw, i.e., "all" (default) for drawing all plot elements, "none" for drawing no horizontal or smoothing curve, "hline" for drawing a horizontal line at 0 using the function ggplot2::geom_hline, "loess" for drawing the LOESS smoothing curve using the function ggplot2::geom_smooth, and "ci.loess" for drawing confidence bands around the LOESS smoothing curve (i.e., se = TRUE).
- hline.type: a numeric value (default: "solid") indicating the linetype argument in the geom_hline function.
- hline.width: a numeric value (default: 1) indicating the linewidth argument in the geom_hline function.
- hline.col: a character string (default: "#FE6100") indicating the color argument in the geom_hline function.
- point.size: a numeric value (default: 2) indicating the size argument in the geom_point function.
- point.shape: a numeric value (default: 21) indicating the shape argument in the geom_point function.
- point.fill: a character string (default: "#648FFF") indicating the fill argument in the geom_point function.
- point.col: a character string (default: "black") indicating the color argument in the geom_point function.
- smooth.fill: a character string (default: "##56B4E9") indicating the fill argument in the geom_smooth function.
- smooth.alpha: a numeric value between 0 and 1 (default: 0.15) indicating the alpha argument in the geom_smooth function.
- smooth.type: a numeric value (default: "dashed") indicating the linetype argument in the geom_smooth function.
- smooth.width: a numeric value (default: 1) indicating the linewidth argument in the geom_smooth function.
- smooth.col: a character string (default: "#0057B8") indicating the color argument in the geom_smooth function.
- xlim: a numeric vector with two elements indicating the limits argument in the scale_x_continuous function.
- xbreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_x_continuous function.
- ylim: a numeric vector with two elements indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.
- axis.title.size: a numeric value (default: 11) indicating the size argument in the element_text function, i.e. theme(axis.title = element_text(size = axis.text.size))
- axis.text.size: a numeric value (default: 10) indicating the size argument in the element_text function, i.e., theme(axis.text = element_text(size = axis.text.size))

Function `test.levene` The `plot` function for the misty object of type `"test.levene"` has following plotting arguments:

- `geom`: a character vector indicating which plot elements to draw, i.e., `"all"` (default) for drawing all plot elements, `"violin"` for drawing violins, `"box"` for drawing boxplots, and `"jitter"` for drawing jittered data points.
- `violin.alpha`: a numeric value between 0 and 1 (default: `0.3`) indicating the `alpha` argument in the `geom_violin` function.
- `violin.trim`: a logical value (default: `FALSE`), indicating the `trim` argument in the `geom_violin` function.
- `box.alpha`: a numeric value between 0 and 1 (default: `0.2`) indicating the `alpha` argument in the `geom_boxplot` function.
- `box.width`: a numeric value between 0 and 1 (default: `0.2`) indicating the `alpha` argument in the `geom_boxplot` function.
- `jitter.size`: a numeric value (default: `1.25`) indicating the `size` argument in the `geom_jitter` function.
- `jitter.width`: a numeric value (default: `0.05`) indicating the `width` argument in the `geom_jitter` function.
- `jitter.height`: a numeric value (default: `0`) indicating the `height` argument in the `geom_jitter` function.
- `jitter.alpha`: a numeric value between 0 and 1 (default: `0.2`) indicating the `alpha` argument in the `geom_jitter` function.
- `gray`: logical: if `TRUE`, the plot is drawn in gray scale.
- `start`: a numeric value between 0 and 1 (default: `0.9`), indicating the `start` argument in the `scale_fill_grey` function.
- `end`: a numeric value between 0 and 1 (default: `0.4`), indicating the `end` argument in the `scale_fill_grey` function.
- `scale.fill.col`: a character vector (default: `NULL`), indicating the `values` argument in the `scale_fill_manual` function.
- `xlab`: a character string (default: `NULL`) indicating the `name` argument in the `scale_x_discrete` function.
- `ylab`: a character string (default: `NULL`) indicating the `name` argument in the `scale_y_continuous` function.
- `ylim`: a numeric vector with two elements (default: `NULL`) indicating the `limits` argument in the `scale_y_continuous` function.
- `ybreaks`: a numeric vector (default: `waiver()`) indicating the `breaks` argument in the `scale_y_continuous` function.
- `title`: a character string (default: `""`) indicating the `title` argument in the `labs` function.
- `subtitle`: a character string (default: `""`) indicating `subtitle` argument in the `labs` function.

Function `test.linear` The `plot` function for the misty object of type `"test.linear"` has following plotting arguments:

- `geom`: a character vector indicating which plot elements to draw, i.e., `"all"` (default) for drawing all plot elements, `"none"` for drawing no regression or smoothing curve, `"lm"` for drawing the regression line, `"loess"` for darwing the LOESS smoothing curve, `"ci.lm"`

for drawing confidence bands around the regression line, "ci.loess" for darwing confidence bands around the LOESS smoothing curve. The default setting is c("lm", "loess", "ci.lm", "ci.loess").

- point.size: a numeric value (default: 2) indicating the size argument in the geom_point function.
- point.shape: a numeric value (default: 21) indicating the shape argument in the geom_point function.
- point.fill: a character string (default: "#648FFF") indicating the fill argument in the geom_point function.
- point.col: a character string (default: "black") indicating the color argument in the geom_point function.
- smooth.fill: a character vector with two elements (default: c("#FE6100", "#56B4E9")) indicating the fill argument in the geom_smooth function for the regression and LOESS smoothing curve.
- smooth.alpha: a numeric vector with two elements between 0 and 1 (default: 0.15) indicating the alpha argument in the geom_smooth function for the regression and LOESS smoothing curve.
- smooth.type: a numeric vector with two elements (default: c("solid", "dashed")) indicating the linetype argument in the geom_smooth function for the regression and LOESS smoothing curve.
- smooth.width: a numeric vector with two elements (default: c(1, 0.8)) indicating the linewidth argument in the geom_smooth function for the regression and LOESS smoothing curve.
- smooth.col: a character vector with two elements (default: c("#FE6100", "#0057B8")) indicating the color argument in the geom_smooth function for the regression and LOESS smoothing curve.
- xlab: a character string (default: "Predictor") indicating the name argument in the scale_x_continuous function.
- xlim: a numeric vector with two elements (default: NULL) indicating the limits argument in the scale_x_continuous function.
- xbreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_x_continuous function.
- ylab: a character string (default: "Partial Residual") indicating the name argument in the scale_y_continuous function.
- ylim: a numeric vector with two elements indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.

Function test.norm The plot function for the misty object of type "test.norm" has following plotting arguments:

- geom: a character vector indicating which plot to draw, i.e., "all" for plotting a Q-Q plot and histogram, "qq" for plotting Q-Q plots, and "hist" for plotting histograms. Note that "all" and c("qq", "hist") are valid only when specifying a fitted model of class lm or a single variable. When specifying "all" or c("qq", "hist") given multiple variables, a multivariate Mahalanobis Q-Q plot is drawn. The default setting is c("qq", "hist").

- ribbon.fill: a character string (default: "#56B4E9") indicating the fill argument in the geom_ribbon function.
- ribbon.alpha: a numeric value between 0 and 1 (default: 0.1) indicating the alpha argument in the geom_ribbon function.
- ribbon.col: a character string (default: "#0057B8") indicating the color argument in the geom_ribbon function.
- abline.col: a character string (default: "#0057B8") indicating the color argument in the geom_abline function.
- abline.width: a numeric value (default: 0.7) indicating the linewidth argument in the geom_abline function.
- point.size: a numeric value (default: 2) indicating the size argument in the geom_point function.
- point.shape: a numeric value (default: 21) indicating the shape argument in the geom_point function.
- point.fill: a character string (default: "#648FFF") indicating the fill argument in the geom_point function.
- point.col: a character string (default: "black") indicating the color argument in the geom_point function.
- hist.binwidth: a numeric value or a function (default: NULL) indicating the binwidth argument in the geom_histogram function.
- hist.fill: a numeric value or a function (default: "gray90") indicating the fill argument in the geom_histogram function.
- hist.col: a numeric value or a function (default: "black") indicating the color argument in the geom_histogram function.
- function.type: a numeric or character vector of length 2 (default: c("solid", "dashed")) indicating the linetype argument in the stat_function function.
- function.width: a numeric vector of length 2 (default: c(1, 1))) indicating the linewidth argument in the stat_function function.
- function.col: a character vector of length 2 (default: c("#0057B8", "#D55E00")) indicating the color argument in the stat_function function.
- xlab: a character string indicating the name argument in the scale_x_continuous function. The default setting is "Theoretical Quantiles of the Normal Distribution" when plotting univariate QQ plots and "Chi-Square Quantile" when plotting a multivariate QQ plot.
- xlim: a numeric vector with two elements indicating the limits argument in the scale_x_continuous function.
- xbreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_x_continuous function.
- ylab: a character string indicating the name argument in the scale_y_continuous function The default setting is "Sample Quantiles" when plotting univariate QQ plots and "Squared Mahalanobis Distance" when plotting a multivariate QQ plot.
- ylim: a numeric vector with two elements indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.

- axis.title.size: a numeric value (default: 11) indicating the size argument in the element_text function, i.e. theme(axis.title = element_text(size = axis.text.size))
- axis.text.size: a numeric value (default: 10) indicating the size argument in the element_text function, i.e., theme(axis.text = element_text(size = axis.text.size))
- facet.nrow: a numeric value (default: NULL) indicating the nrow argument in the facet_wrap function when plotting multiple Q-Q plots or histograms.
- facet.ncol: a numeric value (default: 3) indicating the ncol argument in the facet_wrap function when plotting multiple Q-Q plots or histograms.
- facet.scales: a character string indicating scales argument in the facet_wrap function. The default is "free_y" when plotting multiple Q-Q plots and "free" when plotting multiple histograms.
- strip.text.size: a numeric value (default: 10) indicating the font size of the strip text, i.e. theme(strip.text = ggplot2::element_text(size = strip.text.size)).

Functions test.t**,** test.welch**, and** test.z The plot function for the misty object of type "test.t", "test.welch", and "test.z" has following plotting arguments:

- geom: a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "bar" for drawing bars representing means for each groups, "point" for drawing points representing means for each group, "ci" for drawing error bars representing confidence intervals, "jitter" for drawing jittered data points, "line" for drawing a horizontal line at mu for the one-sample t-test or at 0 for the paired-sample t-test.
- conf.level: a numeric value between 0 and 1 (default: 0.95) indicating the confidence level of the interval.
- adjust: logical: if TRUE (default), difference-adjustment for the confidence intervals in a two-sample design is applied.
- point.size: a numeric value (default: 3) indicating the size argument in the geom_point function.
- errorbar.width: a numeric value (default: 0.1) indicating the width argument in the geom_errorbar function.
- hline.type: an integer value or character string (default: 3) indicating the linetype argument in the geom_hline function.
- hline.width: a numeric value (default: 0.8) indicating the linewidth argument in the geom_hline function.
- jitter.size: a numeric value (default: 1.25) indicating the size argument in the geom_jitter function.
- jitter.width: a numeric value (default: 0.05) indicating the width argument in the geom_jitter function.
- jitter.height: a numeric value (default: 0) indicating the height argument in the geom_jitter function.
- jitter.alpha: a numeric value between 0 and 1 (default: 0.1) the alpha argument in the geom_jitter function.
- xlab: a character string (default: NULL) indicating the name argument in the scale_x_continuous function.
- ylab: a character string (default: "y") indicating the name argument in the scale_y_continuous function.

- ylim: a numeric vector with two elements (default: NULL) indicating the limits argument in the scale_y_continuous function.
- ybreaks: a numeric vector (default: waiver()) indicating the breaks argument in the scale_y_continuous function.
- title: a character string (default: "") indicating the title argument in the labs function.
- subtitle: a character string (default: "") indicating the subtitle argument in the labs function The default setting is "Two-Sided Confidence Interval" when adjust = FALSE or "Two-Sided Difference-Adjusted Confidence Interval" when adjust = TRUE.
- facet_ncol: a numeric value (default: 3) indicating the ncol argument in the facet_wrap function when conducting multiple two-sample t-tests or Welch tests.
- facet_nrow: a numeric value (default: NULL) indicating the nrow argument in the facet_wrap function when conducting multiple two-sample t-tests or Welch tests.
- facet.scales: a character string (default: "free_y") indicating the scales argument in the facet_wrap function when conducting multiple two-sample t-tests or Welch tests.

Author(s)

Takuya Yanagida

---

print.misty.object *Print misty.object object*

Description

This function prints an misty.object object.

Usage

```
## S3 method for class 'misty.object'
print(x, digits = NULL, p.digits = NULL, icc.digits = NULL,
      r.digits = NULL, ess.digits = NULL, mcse.digits = NULL, horiz = TRUE,
      check = TRUE, ...)
```

Arguments

x	misty.object object.
digits	an integer value indicating the number of decimal places to be used for displaying results.
p.digits	an integer indicating the number of decimal places to be used for displaying *p*-values.
icc.digits	an integer indicating the number of decimal places to be used for displaying intraclass correlation coefficients (multilevel.descript() and multilevel.icc() function).

r.digits	an integer value indicating the number of decimal places to be used for displaying R-hat values, item-total correlation standardized factor loadings, coefficient alpha, and coefficient omega.
ess.digits	an integer value indicating the number of decimal places to be used for displaying effective sample sizes.
mcse.digits	an integer value indicating the number of decimal places to be used for displaying Monte Carlo standard errors.
horiz	logical: if TRUE, a horizontal line under the table header is drawn.
check	logical: if TRUE, argument specification is checked.
...	further arguments passed to or from other methods.

Author(s)

Takuya Yanagida

---

read.data *Read Data File in Table Format, SPSS, Excel, or Stata DTA File*

Description

This function reads a (1) data file in CSV (.csv), DAT (.dat), or TXT (.txt) format using the fread function from the **data.table** package, (2) SPSS file (.sav) using the read.sav function, (3) Excel file (.xlsx) using the read.xlsx function, or a (4) Stata DTA file (.dta) using the read.dta function in the **misty** package.

Usage

```
read.data(file, sheet = NULL, header = TRUE, select = NULL, drop = NULL,
          sep = "auto", dec = "auto", use.value.labels = FALSE,
          use.missings = TRUE, na.strings = c("NA", ""),
          stringsAsFactors = FALSE, formats = FALSE, label = FALSE,
          labels = FALSE, missing = FALSE, widths = FALSE, as.data.frame = TRUE,
          encoding = c("unknown", "UTF-8", "Latin-1"), check = TRUE)
```

Arguments

file	a character string indicating the name of the data file with the file extension .csv, .dat, .txt, .sav, .xlsx, or .dta. Note that the function will select an appropriate read-function depending on the file extension.
sheet	a character string indicating the name of a Excel sheet or a numeric value indicating the position of the Excel sheet to read. By default the first sheet will be read when reading an Excel file (.xlsx).
header	logical: if TRUE (default), the first row is used as column names when reading an Excel file (.xlsx), if FALSE default names are used. A character vector giving a name for each column can also be used.

select	a character vector of column names or numeric vector to keep, drop the rest. See the help page of the fread function in the **data.table** package.
drop	a character vector of column names or numeric vector to drop, keep the rest.
sep	a character string indicating the separator between columns for the fread function when reading data in CSV (.csv), DAT (.dat), or TXT (.txt) format.
dec	a character string indicating the decimal separator for the fread function when reading data in CSV (.csv), DAT (.dat), or TXT (.txt) format.
use.value.labels	logical: if TRUE, variables with value labels are converted into factors.
use.missings	logical: if TRUE (default), user-defined missing values are converted into NAs.
na.strings	a character vector of strings which are to be interpreted as NA values.
stringsAsFactors	logical: if TRUE, character vectors are converted to factors.
formats	logical: if TRUE, variable formats are shown in an attribute for all variables.
label	logical: if TRUE, variable labels are shown in an attribute for all variables.
labels	logical: if TRUE, value labels are shown in an attribute for all variables.
missing	logical: if TRUE, value labels for user-defined missings are shown in an attribute for all variables.
widths	logical: if TRUE, widths are shown in an attribute for all variables.
as.data.frame	logical: if TRUE (default), function returns a regular data frame; if FALSE function returns a tibble or data.table.
encoding	a character string indicating the encoding, i.e., "unknown", "UTF-8", or "Latin-1" (default).
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a data frame, tibble, or data table.

Author(s)

Takuya Yanagida

References

Barrett, T., Dowle, M., Srinivasan, A., Gorecki, J., Chirico, M., Hocking, T., & Schwendinger, B. (2024). data.table: Extension of 'data.frame'. R package version 1.16.0. https://CRAN.R-project.org/package=data.table

Wickham H, Miller E, Smith D (2023). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.5.3. https://CRAN.R-project.org/package=haven

See Also

write.data, read.sav, write.sav, write.xlsx, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:

# Example 1: Read CSV data file
dat <- read.data("CSV_Data.csv")

# Example 2: Read DAT data file
dat <- read.data("DAT_Data.dat")

# Example 3: Read TXT data file
dat <- read.data("TXT_Data.txt")

# Example 4: Read SPSS data file
dat <- read.data("SPSS_Data.sav")

# Example 5: Read Excel data file
dat <- read.data("Excel_Data.xlsx")

# Example 6: Read Stata data file
dat <- read.data("Stata_Data.dta")

## End(Not run)
```

`read.dta` *Read Stata DTA File*

Description

This function calls the `read_dta` function in the **haven** package by Hadley Wickham, Evan Miller and Danny Smith (2023) to read a Stata DTA file.

Usage

```
read.dta(file, use.value.labels = FALSE, formats = FALSE, label = FALSE, labels = FALSE,
         missing = FALSE,   widths = FALSE, as.data.frame = TRUE, check = TRUE)
```

Arguments

`file`	a character string indicating the name of the Stata data file with or without file extension '.dta', e.g., "`Stata_Data.dta`" or "`Stata_Data`".
`use.value.labels`	logical: if `TRUE`, variables with value labels are converted into factors.
`formats`	logical: if `TRUE` (default), variable formats are shown in an attribute for all variables.
`label`	logical: if `TRUE`, variable labels are shown in an attribute for all variables.
`labels`	logical: if `TRUE`, value labels are shown in an attribute for all variables.
`missing`	logical: if `TRUE`, convert tagged missing values to regular R `NA`.

`widths`	logical: if `TRUE`, widths are shown in an attribute for all variables.
`as.data.frame`	logical: if `TRUE` (default), function returns a regular data frame; if `FALSE` function returns a tibble.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns a data frame or tibble.

Note

This function is a modified copy of the `read_dta()` function in the **haven** package by Hadley Wickham, Evan Miller and Danny Smith (2023).

Author(s)

Takuya Yanagida

References

Wickham H, Miller E, Smith D (2023). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.5.3. https://CRAN.R-project.org/package=haven

See Also

`read.data`, `write.data`, `read.sav`, `write.sav`, `write.xlsx`, `write.dta`, `read.mplus`, `write.mplus`

Examples

```
## Not run:

read.dta("Stata_Data.dta")
read.dta("Stata_Data")

# Example 2: Read Stata data, convert variables with value labels into factors
read.dta("Stata_Data.dta", use.value.labels = TRUE)

# Example 3: Read Stata data as tibble
read.dta("Stata_Data.dta", as.data.frame = FALSE)

## End(Not run)
```

read.mplus *Read Mplus Data File and Variable Names*

Description

This function reads a Mplus data file and/or Mplus input/output file to return a data frame with variable names extracted from the Mplus input/output file. Note that by default `-99` in the Mplus data file is replaced with to `NA`.

Usage

```
read.mplus(file, sep = "", input = NULL, na = -99, print = FALSE, return.var = FALSE,
           encoding = "UTF-8-BOM", check = TRUE)
```

Arguments

`file`	a character string indicating the name of the Mplus data file with or without the file extension `.dat`, e.g., `"Mplus_Data.dat"` or `"Mplus_Data"`. Note that it is not necessary to specify this argument when `return.var = TRUE`.
`sep`	a character string indicating the field separator (i.e., delimiter) used in the data file specified in `file`. By default, the separator is 'white space', i.e., one or more spaces, tabs, newlines or carriage returns.
`input`	a character string indicating the Mplus input (`.inp`) or output file (`.out`) in which the variable names are specified in the `VARIABLE:` section. Note that if `input = NULL`, this function is equivalent to `read.table(file)`.
`na`	a numeric vector indicating values to replace with `NA`. By default, `-99` is replaced with `NA`. If `-99` is not a missing value change the argument to `NULL`.
`print`	logical: if `TRUE`, variable names are printed on the console.
`return.var`	logical: if `TRUE`, the function returns the variable names extracted from the Mplus input or output file only.
`encoding`	character string declaring the encoding used on `file` so the character data can be re-encoded.See the 'Encoding' section of the help page for the `file` function, the 'R Data Import/Export Manual' and 'Note'.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

A data frame containing a representation of the data in the file.

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.data, write.data, read.sav, write.sav, write.xlsx, read.dta, write.dta, write.mplus

Examples

```
## Not run:

# Example 1: Read Mplus data file and variable names extracted from the Mplus input file
dat <- read.mplus("Mplus_Data.dat", input = "Mplus_Input.inp")

# Example 2: Read Mplus data file and variable names extracted from the Mplus input file,
# print variable names on the console
dat <- read.mplus("Mplus_Data.dat", input = "Mplus_Input.inp", print = TRUE)

# Example 3: Read variable names extracted from the Mplus input file
varnames <- read.mplus(input = "Mplus_Input.inp", return.var = TRUE)

## End(Not run)
```

---

read.sav *Read SPSS File*

Description

This function calls the read_spss function in the **haven** package by Hadley Wickham, Evan Miller and Danny Smith (2023) to read an SPSS file.

Usage

```
read.sav(file, use.value.labels = FALSE, use.missings = TRUE, formats = FALSE,
         label = FALSE, labels = FALSE, missing = FALSE, widths = FALSE,
         as.data.frame = TRUE, check = TRUE)
```

Arguments

file	a character string indicating the name of the SPSS data file with or without file extension '.sav', e.g., "SPSS_Data.sav" or "SPSS_Data".
use.value.labels	logical: if TRUE, variables with value labels are converted into factors.
use.missings	logical: if TRUE (default), user-defined missing values are converted into NAs.
formats	logical: if TRUE, variable formats are shown in an attribute for all variables.
label	logical: if TRUE, variable labels are shown in an attribute for all variables.
labels	logical: if TRUE, value labels are shown in an attribute for all variables.
missing	logical: if TRUE, value labels for user-defined missings are shown in an attribute for all variables.
widths	logical: if TRUE, widths are shown in an attribute for all variables.

as.data.frame	logical: if TRUE (default), function returns a regular data frame; if FALSE function returns a tibble.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a data frame or tibble.

Author(s)

Takuya Yanagida

References

Wickham H, Miller E, & Smith D (2023). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.5.3. https://CRAN.R-project.org/package=haven

See Also

read.data, write.data, write.sav, write.xlsx, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:

# Example 1: Read SPSS data file
read.sav("SPSS_Data.sav")
read.sav("SPSS_Data")

# Example 2: Read SPSS data file, convert variables with value labels into factors
read.sav("SPSS_Data.sav", use.value.labels = TRUE)

# Example 3: Read SPSS data file, user-defined missing values are not converted into NAs
read.sav("SPSS_Data.sav", use.missing = FALSE)

# Example 4: Read SPSS data file as tibble
read.sav("SPSS_Data.sav", as.data.frame = FALSE)

## End(Not run)
```

---

read.xlsx *Read Excel File*

Description

This function calls the read_xlsx() function in the **readxl** package by Hadley Wickham and Jennifer Bryan (2019) to read an Excel file (.xlsx).

Usage

```
read.xlsx(file, sheet = NULL, header = TRUE, range = NULL,
       coltypes = c("skip", "guess", "logical", "numeric", "date", "text", "list"),
          na = "", trim = TRUE, skip = 0, nmax = Inf, guessmax = min(1000, nmax),
           progress = readxl::readxl_progress(), name.repair = "unique",
           as.data.frame = TRUE, check = TRUE)
```

Arguments

file	a character string indicating the name of the Excel data file with or without file extension '.xlsx', e.g., "My_Excel_Data.xlsx" or "My_Excel_Data".
sheet	a character string indicating the name of a sheet or a numeric value indicating the position of the sheet to read. By default the first sheet will be read.
header	logical: if TRUE (default), the first row is used as column names, if FALSE default names are used. A character vector giving a name for each column can also be used. If coltypes as a vector is provided, colnames can have one entry per column, i.e. have the same length as coltypes, or one entry per unskipped column.
range	a character string indicating the cell range to read from, e.g. typical Excel ranges like "B3:D87", possibly including the sheet name like "Data!B2:G14". Interpreted strictly, even if the range forces the inclusion of leading or trailing empty rows or columns. Takes precedence over skip, nmax and sheet.
coltypes	a character vector containing one entry per column from these options "skip", "guess", "logical", "numeric", "date", "text" or "list". If exactly one coltype is specified, it will be recycled. By default (i.e., coltypes = NULL) coltypes will be guessed. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from coltypes = NULL, but on a cell-by-cell basis.
na	a character vector indicating strings to interpret as missing values. By default, blank cells will be treated as missing data.
trim	logical: if TRUE (default), leading and trailing whitespace will be trimmed.
skip	a numeric value indicating the minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if the argument range is specified.
nmax	a numeric value indicating the maximum number of data rows to read. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the returned data frame. Ignored if the argument range is specified.
guessmax	a numeric value indicating the maximum number of data rows to use for guessing column types.
progress	display a progress spinner? By default, the spinner appears only in an interactive session, outside the context of knitting a document, and when the call is likely to run for several seconds or more.
name.repair	a character string indicating the handling of column names. By default, the function ensures column names are not empty and are unique.

as.data.frame	logical: if TRUE (default), function returns a regular data frame; if FALSE function returns a tibble.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a data frame or tibble.

Author(s)

Takuya Yanagida

References

Wickham H, Miller E, Smith D (2023). *readxl: Read Excel Files*. R package version 1.4.3. https://CRAN.R-project.org/package=readxl

See Also

read.data, write.data, read.sav, write.sav, write.xlsx, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:

# Example 1: Read Excel file (.xlsx)
read.xlsx("data.xlsx")

# Example 1: Read Excel file (.xlsx), use default names as column names
read.xlsx("data.xlsx", header = FALSE)

# Example 2: Read Excel file (.xlsx), interpret -99 as missing values
read.xlsx("data.xlsx", na = "-99")

# Example 3: Read Excel file (.xlsx), use x1, x2, and x3 as column names
read.xlsx("data.xlsx", header = c("x1", "x2", "x3"))

# Example 4: Read Excel file (.xlsx), read cells A1:B5
read.xlsx("data.xlsx", range = "A1:B5")

# Example 5: Read Excel file (.xlsx), skip 2 rows before reading data
read.xlsx("data.xlsx", skip = 2)

# Example 5: Read Excel file (.xlsx), return a tibble
read.xlsx("data.xlsx", as.data.frame = FALSE)

## End(Not run)
```

rec *Recode Variable*

Description

This function recodes numeric vectors, character vectors, or factors according to recode specifications.

Usage

```
rec(data, ..., spec, as.factor = FALSE, levels = NULL, append = TRUE,
    name = ".e", as.na = NULL, table = FALSE, check = TRUE)
```

Arguments

data	a numeric vector, character vector, factor, or data frame.
...	an expression indicating the variable names in data, e.g., rec(dat, x1, x2, x3, spec = "1 = 0")). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
spec	a character string of recode specifications (see 'Details').
as.factor	logical: if TRUE, character vector will be coerced to a factor.
levels	a character vector for specifying the levels in the returned factor.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
append	logical: if TRUE (default), centered variable(s) are appended to the data frame specified in the argument data.
name	a character vector indicating the names of the recoded variables. By default, variables are named with the ending ".r" Resulting in e.g. "x1.r" and "x2.r". Variable names can also be specified using a character vector matching the number of variables specified in data (e.g., name = c("recode.x1", "recode.x2")).
table	logical: if TRUE, a cross table variable x recoded variable is printed on the console if only one variable is specified in data.
check	logical: if TRUE (default), argument specification is checked.

Details

Recode specifications appear in a character string, separated by semicolons (see the examples below), of the form input = output. If an input value satisfies more than one specification, then the first (from left to right) applies. If no specification is satisfied, then the input value is carried over to the result. NA is allowed in input and output. Several recode specifications are supported:

Single Value For example, spec = "0 = NA".

Vector of Values For example, spec = "c(7, 8, 9) = 'high'".

Range of Values For example, `spec = "7:9 = 'C'"`. The special values `lo` (lowest value) and `hi` (highest value) may appear in a range. For example, `spec = "lo:10 = 1"`. Note that `:` is not the R sequence operator. In addition you may not use `:` with the collect operator, e.g., `spec = "c(1, 3, 5:7)"` will cause an error.

else For example, `spec = "0 = 1; else = NA"`. Everything that does not fit a previous specification. Note that `else` matches all otherwise unspecified values on input, including `NA`.

Value

Returns a numeric vector or data frame with the same length or same number of rows as `data` containing the recoded coded variable(s).

Note

This function was adapted from the `recode()` function in the **car** package by John Fox and Sanford Weisberg (2019).

Author(s)

Takuya Yanagida

References

Fox, J., & Weisberg S. (2019). *An R Companion to Applied Regression* (3rd ed.). Thousand Oaks CA: Sage. URL: https://socialsciences.mcmaster.ca/jfox/Books/Companion/

See Also

`coding`, `item.reverse`

Examples

```
#————————————————————————————————————————————————————————————————————————
# Example 1: Numeric vector
x.num <- c(1, 2, 4, 5, 6, 8, 12, 15, 19, 20)

# Example 1a: Recode 5 = 50 and 19 = 190
rec(x.num, spec = "5 = 50; 19 = 190")

# Example 1b: Recode 1, 2, and 5 = 100 and 4, 6, and 7 = 200 and else = 300
rec(x.num, spec = "c(1, 2, 5) = 100; c(4, 6, 7) = 200; else = 300")

# Example 1c: Recode lowest value to 10 = 100 and 11 to highest value = 200
rec(x.num, spec = "lo:10 = 100; 11:hi = 200")

# Example 1d: Recode 5 = 50 and 19 = 190 and check recoding
rec(x.num, spec = "5 = 50; 19 = 190", table = TRUE)

#————————————————————————————————————————————————————————————————————————
# Example 2: Character vector
x.chr <- c("a", "c", "f", "j", "k")
```

```
# Example 2a: Recode a to x
rec(x.chr, spec = "'a' = 'X'")

# Example 2b: Recode a and f to x, c and j to y, and else to z
rec(x.chr, spec = "c('a', 'f') = 'x'; c('c', 'j') = 'y'; else = 'z'")

# Example 2c: Recode a to x and coerce to a factor
rec(x.chr, spec = "'a' = 'X'", as.factor = TRUE)

#-----------------------------------------------------------------------
# Example 3: Factor
x.fac <- factor(c("a", "b", "a", "c", "d", "d", "b", "b", "a"))

# Example 3a: Recode a to x, factor levels ordered alphabetically
rec(x.fac, spec = "'a' = 'x'")

# Example 3b: Recode a to x, user-defined factor levels
rec(x.fac, spec = "'a' = 'x'", levels = c("x", "b", "c", "d"))

#-----------------------------------------------------------------------
# Example 4: Multiple variables
dat <- data.frame(x1.num = c(1, 2, 4, 5, 6),
                  x2.num = c(5, 19, 2, 6, 3),
                  x1.chr = c("a", "c", "f", "j", "k"),
                  x2.chr = c("b", "c", "a", "d", "k"),
                  x1.fac = factor(c("a", "b", "a", "c", "d")),
                  x2.fac = factor(c("b", "a", "d", "c", "e")))

# Example 4a: Recode numeric vector and attach to 'dat'
cbind(dat, rec(dat[, c("x1.num", "x2.num")], spec = "5 = 50; 19 = 190"))

# Alternative specification using the 'data' argument,
rec(dat, x1.num, x2.num, spec = "5 = 50; 19 = 190")

# Example 4b: Recode character vector and attach to 'dat'
cbind(dat, rec(dat[, c("x1.chr", "x2.chr")], spec = "'a' = 'X'"))

# Example 4c: Recode factor vector and attach to 'dat'
cbind(dat, rec(dat[, c("x1.fac", "x2.fac")], spec = "'a' = 'X'"))
```

`restart` *Restart R Session*

Description

This function restarts the RStudio session and is equivalent to using the menu item `Session - Restart R`.

Usage

```
restart()
```

Details

The function call `executeCommand("restartR")` in the package **rstudioapi** is used to restart the R session. Note that the function `restartSession()` in the package **rstudioapi** is not equivalent to the menu item `Session - Restart R` since it does not unload packages loaded during an R session.

Author(s)

Takuya Yanagida

References

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2022). rstudioapi: Safely access the RStudio API. R package version 0.14. https://CRAN.R-project.org/package=rstudioapi

Examples

```
## Not run:

# Example 1: Restart the R Session
restart()

## End(Not run)
```

`robust.lmer` *Robust Estimation of Multilevel and Linear Mixed-Effects Models*

Description

This function estimates a multilevel and linear mixed-effects model based on a robust estimation method using the `rlmer()` function from the **robustlmm** package that down-weights observations depending on robustness weights computed by robustification of the scoring equations and an application of the Design Adaptive Scale (DAS) approach.

Usage

```
robust.lmer(model, method = c("DAStau", "DASvar"),setting = c("RSEn", "RSEa"),
            digits = 2, p.digits = 3, write = NULL, append = TRUE, check = TRUE,
            output = TRUE)
```

Arguments

`model`	a fitted model of class `"lmerMod"` or `"lmerModLmerTest"`.
`method`	a character string indicating the method used for estimating theta and sigma, i.e., `"DAStau"` (default) for using numerical quadrature for computing the consistency factors and `"DASvar"` for computing the consistency factors using a direct approximation. Note that `"DAStau"` is slower than `"DASvar"` but yields more accurate results. However, complex models with correlated random effects with more than one correlation term can only estimated using `"DASvar"`. See help page of the `rlmer()` function in the R package `robustlmm` for more details.

setting	a character string indicating the setting for the parameters used for computing the robustness weights, i.e., "RSEn" (default) and "RSEa" for higher asymptotic efficiency which results in lower robustness. See help page of the rlmer() function in the R package robustlmm for more details.
digits	an integer value indicating the number of decimal places to be used.
p.digits	an integer value indicating the number of decimal places to be used for displaying *p*-value.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Function specification and Function Arguments The function rlmer from the **robustlmm** package is specified much like the function lmer from the lme4 package, i.e., a formula object and data frame is specified as the first and second argument. However, the robust.lmer function requires a fitted "lmerMod" or "lmerModLmerTest" that is used to re-estimate the model using the robust estimation method. Note that the function rlmer provides the additional arguments rho.e, rho.b, rho.sigma.e, rho.sigma.b, rel.tol, max.iter, verbose, doFit, init, and initTheta that are not supported by the robust.lmer function. See help page of the rlmer() function in the R package robustlmm for more details.

Statistical Significance Testing The function rlmer from the **robustlmm** package does not provide any degrees of freedom or significance values. When specifying a "lmerModLmerTest" object for the argument model, the robust.lmer function uses the Satterthwaite or Kenward-Roger degrees of freedom from the "lmerModLmerTest" object to compute significance values for the regression coefficients based on parameter estimates and standard error of the robust multilevel mixed-effects (see Sleegers et al. (2021).

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
model	object returned from the rlmer function
args	specification of function arguments
result	list with results, i.e., call for the the function call, randeff for the variance and correlation components, coef for the model coefficients, weights for the robustness weights, and and converg for the convergence check, i.e., 1 = model converged, 0 = model singular, and -1 model not converged.

Author(s)

Takuya Yanagida

References

Koller, M. (2016). robustlmm: An R Package for Robust Estimation of Linear Mixed-Effects Models. *Journal of Statistical Software, 75*(6), 1–24. https://doi.org/10.18637/jss.v075.i06

See Also

coeff.robust, summa

Examples

```
## Not run:
# Load lme4, lmerTest, and misty package
misty::libraries(lme4, lmerTest, misty)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#————————————————————————————————————————————————————————————————————————
# Multilevel and Linear Mixed-Effects Model

# Cluster-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Grand-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, w1, type = "CGM", cluster = "cluster")

# Estimate two-level mixed-effects model
mod.lmer2 <- lmer(y1 ~ x2.c + w1.c + (1 | cluster), data = Demo.twolevel)

# Example 1a: Default setting
mod.lmer2r <- robust.lmer(mod.lmer2)

# Example 1b: Extract robustness weights
mod.lmer2r$result$weight$iresid
mod.lmer2r$result$weight$iranef

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 2a: Write results into a text file
robust.lmer(mod.lmer2, write = "Robust_lmer.txt", output = FALSE)

# Example 2b: Write results into a Excel file
robust.lmer(mod.lmer2, write = "Robust_lmer.xlsx", output = FALSE)

## End(Not run)
```

script.copy *Save Copy of the Current Script in RStudio*

Description

This function saves a copy of the current script in RStudio. By default, a folder called `_R_Script_Archive` will be created to save the copy of the current R script with the current date and time into the folder. Note that the current R script needs to have a file location before the script can be copied.

Usage

```
script.copy(file = NULL, folder = "_R_Script_Archive", create.folder = TRUE,
            time = TRUE, format = "%Y-%m-%d_%H%M", overwrite = TRUE,
            check = TRUE)
```

Arguments

`file`	a character string naming the file of the copy without the file extension `".R"`. By default, the file of the copy has the same name as the original file.
`folder`	a character string naming the folder in which the file of the copy is saved. If `NULL`, the file of the copy is saved in the same folder as the original file. By default, the file of the copy is saved into a folder called `"_R_Script_Archive"`.
`create.folder`	logical: if `TRUE` (default), folder(s) specified in the `file` argument is created. If `FALSE` and the folder does not exist, then a error message is printed on the console.
`time`	logical: if `TRUE` (default), the current time is attached to the name of the file specified in the argument `file`.
`format`	a character string indicating the format if the `POSIXct` class resulting from the `Sys.time` function. The default setting provides a character string indicating the year, month, day, minutes, and seconds. See the help page of the `format.POSIXct` function.
`overwrite`	logical: if `TRUE` (default) an existing destination file is overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.

Note

This function uses the `getSourceEditorContext()` function in the **rstudioapi** package by Kevin Ushey, JJ Allaire, Hadley Wickham, and Gary Ritchie (2023).

Author(s)

Takuya Yanagida

References

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2023). *rstudioapi: Safely access the RStudio API*. R package version 0.15.0 https://CRAN.R-project.org/package=rstudioapi

See Also

script.new, script.close, script.open, script.save, setsource

Examples

```
## Not run:

# Example 1: Save copy current R script into the folder '_R_Script_Archive'
script.copy()

# Exmample 2: Save current R script as 'R_Script.R' into the folder 'Archive'
script.copy("R_Script", folder = "Archive", time = FALSE)

## End(Not run)
```

---

script.new *Open new R Script, R Markdown script, or SQL Script in RStudio*

Description

This function opens a new R script, R markdown script, or SQL script in RStudio.

Usage

```
script.new(text = "", type = c("r", "rmarkdown", "sql"),
           position = rstudioapi::document_position(0, 0),
           run = FALSE, check = TRUE)
```

Arguments

text	a character vector indicating what text should be inserted in the new R script. By default, an empty script is opened.
type	a character string indicating the type of document to be created, i.e., r (default) for an R script, rmakrdown for an R Markdown file, or sql for an SQL script.
position	document_position() function in the **rstudioapi** package indicating the cursor position.
run	logical: if TRUE, the code is executed after the document is created.
check	logical: if TRUE (default), argument specification is checked.

Note

This function uses the documentNew() function in the **rstudioapi** package by Kevin Ushey, JJ Allaire, Hadley Wickham, and Gary Ritchie (2023).

Author(s)

Takuya Yanagida

References

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2023). *rstudioapi: Safely access the RStudio API*. R package version 0.15.0 https://CRAN.R-project.org/package=rstudioapi

See Also

`script.close`, `script.open`, `script.save`, `script.copy`, `setsource`

Examples

```
## Not run:

# Example 1: Open new R script file
script.new()

# Example 2: Open new R script file and run some code
script.new("#----------------------------
# Example

# Generate 100 random numbers
rnorm(100)")

## End(Not run)
```

---

script.open — *Open, Close and Save R Script in RStudio*

Description

The function `script.open` opens an R script, R markdown script, or SQL script in RStudio, the function `script.close` closes an R script, and the function `script.save` saves an R script. Note that the R script need to have a file location before the script can be saved.

Usage

```
script.open(path, line = 1, col = 1, cursor = TRUE, run = FALSE,
            echo = TRUE, max.length = 999, spaced = TRUE, check = TRUE)

script.close(save = FALSE, check = TRUE)

script.save(all = FALSE, check = TRUE)
```

Arguments

`path`	a character string indicating the path of the script.
`line`	a numeric value indicating the line in the script to navigate to.
`col`	a numeric value indicating the column in the script to navigate to.

cursor	logical: if TRUE (default), the cursor moves to the requested location after opening the document.
run	logical: if TRUE, the code is executed after the document is opened
echo	logical: if TRUE (default), each expression is printed after parsing, before evaluation.
max.length	a numeric value indicating the maximal number of characters output for the deparse of a single expression.
spaced	logical: if TRUE (default), empty line is printed before each expression.
save	logical: if TRUE, the script is saved before closing when using the function script.close.
all	logical: if TRUE, all scripts opened in RStudio are saved when using the function script.save.
check	logical: if TRUE (default), argument specification is checked.

Note

This function uses the documentOpen(), documentPath(), documentClose(), documentSave(), and documentSaveAll() functions in the **rstudioapi** package by Kevin Ushey, JJ Allaire, Hadley Wickham, and Gary Ritchie (2023).

Author(s)

Takuya Yanagida

References

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2023). *rstudioapi: Safely access the RStudio API*. R package version 0.15.0 https://CRAN.R-project.org/package=rstudioapi

See Also

script.save, script.copy, setsource

Examples

```
## Not run:

# Example 1: Open  R script file
script.open("script.R")

# Example 2: Open  R script file and run the code
script.open("script.R", run = TRUE)

# Example 3: Close current R script file
script.close()

# Example 4: Save current R script
script.save()
```

```
# Example 5: Save all R scripts
script.save(all = TRUE)

## End(Not run)
```

---

setsource *Set Working Directory to the Source File Location*

Description

This function sets the working directory to the source file location (i.e., path of the current R script) in RStudio and is equivalent to using the menu item `Session - Set Working Directory - To Source File Location`. Note that the R script needs to have a file location before this function can be used.

Usage

```
setsource(path = TRUE, check = TRUE)
```

Arguments

`path`	logical: if `TRUE` (default), the path of the source file is shown on the console.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns the path of the source file location.

Note

This function uses the `documentPath()` function in the **rstudioapi** package by Kevin Ushey, JJ Allaire, Hadley Wickham, and Gary Ritchie (2023).

Author(s)

Takuya Yanagida

References

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2023). *rstudioapi: Safely access the RStudio API*. R package version 0.15.0 https://CRAN.R-project.org/package=rstudioapi

See Also

`script.close`, `script.new`, `script.open`, `script.save`

Examples

```
## Not run:

# Example 1: Set working directory to the source file location
setsource()

# Example 2: Set working directory to the source file location
# and assign path to an object
path <- setsource()
path

## End(Not run)
```

`sim.lavaan` *Generates Simulated Data from a lavaan Model Syntax*

Description

This function simulates data from a lavaan model syntax with unstandardized or standardized parameters. By default, the function simulates observed variables based on the model specified in the argument `model` in a unstandardized metric.

Usage

```
sim.lavaan(model, n = 500, std = FALSE, skew = NULL, kurt = NULL,
           observed = TRUE, latent = FALSE, fscores = FALSE, composites = FALSE,
           errors = FALSE, matrices = FALSE, method = c("eigen", "svd", "chol"),
            seed = NULL, max.iter = 100, check = TRUE)
```

Arguments

`model`	a character string indicating the lavaan model syntax.
`n`	a numeric value indicating the number of observations. By default, 500 simulated cases are simulated.
`std`	logical: if `TRUE`, lavaan model syntax specified in the argument `model` is based on standardized parameters.
`skew`	a numeric vector indicating the skewness values for the observed variables. Note that this argument is only used when `std = FALSE`.
`kurt`	a numeric vector indicating the kurtosis values for the observed variables. Note that this argument is only used when `std = FALSE`. Note that this argument is only used when `std = FALSE`.
`observed`	logical: if `TRUE` (default), observed variables are included. Note that this argument is only used when `std = TRUE`.
`latent`	logical: if `TRUE`, latent variables are included. Note that this argument is only used when `std = TRUE`.

`fscores`	logical: if `TRUE`, factor score are included. Note that this argument is only used when `std = TRUE`.
`composites`	logical: if `TRUE`, composite variables are included. Note that this argument is only used when `std = TRUE`.
`errors`	logical: if `TRUE`, observed error and latent disturbance variables are included. Note that this argument is only used when `std = TRUE`.
`matrices`	logical: if `TRUE`, matrices are included as attributes. Note that this argument is only used when `std = TRUE`.
`method`	a character string indicating the matrix decomposition used to determine the matrix root of `sigma` in the random number generator for the multivariate normal distribution, i.e., `"eigen"` (default) for eigenvalue decomposition, `"svd"` for singular value decomposition, and `"chol"` for Cholesky decomposition. Note that this argument is only used when `std = TRUE`.
`seed`	a numeric value specifying the seed of the pseudo-random numbers used when simulating multivariate normal data.
`max.iter`	a numeric value indicating the maximum number of iterations when solving for error variances and correlation matrix. Note that this argument is only used when `std = TRUE`.
`check`	logical: if `TRUE` (default), argument specification is checked.

Value

Returns a data frame.

Note

This function uses the function `simulateData` from the R package **lavaan** by Yves Rosseel (2012) when `std = FALSE` and is based on modified copies of the function `sim_standardized` from the **simstandard** package by W. Joel Schneider (2021) and the function `rmvnorm` from the package **mvtnorm** by Alan Genz and Frank Bretz (2026) when `std = TRUE`.

Author(s)

Takuya Yanagida

References

Genz, A., & Bretz, F. (2026). *mvtnorm: Multivariate Normal and t Distributions*. R package version 1.3-6. https://doi.org/10.32614/CRAN.package.mvtnorm

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software, 48*, 1-36. https://doi.org/10.18637/jss.v048.i02

Schneider, W. J. (2021). *simstandard: Generate standardized data*. R package version 0.6.3. https://doi.org/10.32614/CRAN.package.simstandard

Examples

```
## Not run:

# Model specification
model <- '# Measurement model
          f1 =~ 0.8*x1 + 0.7*x2 + 0.5*x3
          f2 =~ 0.7*x4 + 0.8*x5 + 0.6*x6
          # Factor correlation
          f1 ~~ 0.4*f2'

# Example 1: Unstandardized parameters, simulate 200 cases
simdat1 <- sim.lavaan(model, n = 200)

# Example 2: Standardized parameters, simulate 200 cases
simdat2 <- sim.lavaan(model, std = TRUE, n = 200)

## End(Not run)
```

`size.mean` *Sample Size Determination*

Description

This function performs sample size determination the one-sample and two-sample t-tests, proportions, and Pearson product-moment correlation coefficients based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

Usage

```
size.mean(delta, sample = c("two.sample", "one.sample"),
          alternative = c("two.sided", "less", "greater"),
          alpha = 0.05, beta = 0.1, write = NULL, append = TRUE,
          check = TRUE, output = TRUE)

size.prop(pi = 0.5, delta, sample = c("two.sample", "one.sample"),
          alternative = c("two.sided", "less", "greater"),
          alpha = 0.05, beta = 0.1, correct = FALSE, write = NULL,
          append = TRUE, check = TRUE, output = TRUE)

size.cor(rho, delta,
         alternative = c("two.sided", "less", "greater"),
         alpha = 0.05, beta = 0.1, write = NULL, append = TRUE,
         check = TRUE, output = TRUE)
```

Arguments

`delta` a numeric value indicating the minimum mean difference to be detected, δ.

sample	a character string specified in the function size.mean or size.prop specifying a one- or two-sample t-test or a proportion test, i.e., "two.sample" (default) for a two-sample test and "one.sample" for a one-sample test.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
alpha	a numeric value indicating the type-I-risk, α.
beta	a numeric value indicating the type-II-risk, β.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.
pi	a numeric value specified in the function size.prop indicating the true value of the probability under the null hypothesis in the one-sample test $\pi.0$ or a number indicating the true value of the probability in group 1 in the two-sample test $\pi.1$.
rho	a numeric value specified in the function size.cor indicating the correlation coefficient under the null hypothesis $\rho.0$.
correct	logical: if TRUE, continuity correction is applied.

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Rasch, D., Pilz, J., Verdooren, L. R., & Gebhardt, G. (2011). *Optimal experimental design with R*.Chapman & Hall/CRC.

See Also

test.t, prop.test, cor.test, cor.matrix

Examples

```
#————————————————————————————————————————————————————————————————————
# One- and Two-Sample t-Test

# Example 1a: One-sample t-test
# H0: mu = mu.0, H1: mu != mu.0
# alpha = 0.05, beta = 0.2, delta = 0.5
size.mean(delta = 0.5, sample = "one.sample",
          alternative = "two.sided", alpha = 0.05, beta = 0.2)

# Example 1b: One-sided two-sample test
```

```
# H0: mu.1 >= mu.2, H1: mu.1 < mu.2
# alpha = 0.01, beta = 0.1, delta = 1
size.mean(delta = 1, sample = "two.sample",
          alternative = "less", alpha = 0.01, beta = 0.1)

#------------------------------------------------------------------------
# One- and Two-Sample Test for Proportions

# Example 2a: Two-sided one-sample test
# H0: pi = 0.5, H1: pi != 0.5
# alpha = 0.05, beta = 0.2, delta = 0.2
size.prop(pi = 0.5, delta = 0.2, sample = "one.sample",
          alternative = "two.sided", alpha = 0.05, beta = 0.2)

# Example 2b: One-sided two-sample test
# H0: pi.1 <=  pi.1 = 0.5, H1: pi.1 > pi.2
# alpha = 0.01, beta = 0.1, delta = 0.2
size.prop(pi = 0.5, delta = 0.2, sample = "two.sample",
          alternative = "greater", alpha = 0.01, beta = 0.1)

#------------------------------------------------------------------------
# Testing the Pearson Product-Moment Correlation Coefficient

# H0: rho = 0.3, H1: rho != 0.3
# alpha = 0.05, beta = 0.2, delta = 0.2
size.cor(rho = 0.3, delta = 0.2, alpha = 0.05, beta = 0.2)

# H0: rho <= 0.3, H1: rho > 0.3
# alpha = 0.05, beta = 0.2, delta = 0.2
size.cor(rho = 0.3, delta = 0.2,
         alternative = "greater", alpha = 0.05, beta = 0.2)
```

---

skewness *Univariate and Multivariate Skewness and Kurtosis*

Description

The function skewness computes the univariate sample or population skewness and conduct's Mardia's test for multivariate skewness, while the function kurtosis computes the univariate sample or population (excess) kurtosis or the multivariate (excess) kurtosis and conduct's Mardia's test for multivariate kurtosis. By default, the function computes the sample univariate skewness or multivariate skewness and the univariate sample excess kurtosis or multivariate excess kurtosis.

Usage

```
skewness(data, ..., sample = FALSE, digits = 2, p.digits,
         as.na = NULL, check = TRUE, output = TRUE)

kurtosis(data, ..., sample = FALSE, center = TRUE, digits = 2, p.digits,
         as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`data`	a numeric vector or data frame.
`...`	an expression indicating the variable names in `data`, e.g., `skewness(dat, x1)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`sample`	logical: if `TRUE`, the univariate sample skewness or kurtosis is computed, while the population skewness or kurtosis is computed when `sample = FALSE`.
`center`	logical: if `TRUE` (default), the univariate or multivariate kurtosis is centered, so that the expected kurtosis under univariate or multivariate normality is 0, while the expected kurtosis under univariate or multivariate normality is 3 when `center = FALSE`.
`digits`	an integer value indicating the number of decimal places to be used. Note that this argument only applied when computing multivariate skewness and kurtosis.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-values.
`as.na`	a numeric vector indicating user-defined missing values, i.e., these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console. Note that this argument only applied when computing multivariate skewness and kurtosis.

Details

Univariate Skewness and Kurtosis Univariate skewness and kurtosis are computed based on the same formula as in SAS and SPSS:

- *Population Skewness*

$$\sqrt{n}\frac{\sum_{i=1}^{n}(X_i-\bar{X})^3}{(\sum_{i=1}^{n}(X_i-\bar{X})^2)^{3/2}}$$

- *Sample Skewness*

$$\frac{n\sqrt{n-1}}{n-2}\frac{\sum_{i=1}^{n}(X_i-\bar{X})^3}{(\sum_{i=1}^{n}(X_i-\bar{X})^2)^{3/2}}$$

- *Population Excess Kurtosis*

$$n\frac{\sum_{i=1}^{n}(X_i-\bar{X})^4}{(\sum_{i=1}^{n}(X_i-\bar{X})^2)^{2}}-3$$

- *Sample Excess Kurtosis*

$$(n+1)\frac{\sum_{i=1}^{n}(X_i-\bar{X})^4}{(\sum_{i=1}^{n}(X_i-\bar{X})^2)^{2}}-3+6\frac{n-1}{(n-2)(n-3)}$$

Note that missing values (`NA`) are stripped before the computation and that at least 3 observations are needed to compute skewness and at least 4 observations are needed to compute kurtosis.

Multivariate Skewness and Kurtosis Mardia's multivariate skewness and kurtosis compares the joint distribution of several variables against a multivariate normal distribution. The expected skewness is 0 for a multivariate normal distribution, while the expected kurtosis is $p(p+2)$ for a multivariate distribution of p variables. However, this function scales the multivariate kurtosis on $p(p+2)$ according to the default setting `center = TRUE` so that the expected kurtosis under multivariate normality is 0. Multivariate skewness and kurtosis are tested for statistical significance based on the chi-square distribution for skewness and standard normal distribution for the kurtosis. If at least one of the tests is statistically significant, the underlying joint population is inferred to be non-normal. Note that non-significance of these statistical tests do not imply multivariate normality.

Value

Returns univariate skewness or kurtosis of `data` or an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`data`	a numeric vector or data frame specified in `data`
`args`	specification of function arguments
`result`	result table

Note

These functions implemented a modified copy of the `mardia()` function in the **psych** package by William Revelle (2024).

Author(s)

Takuya Yanagida

References

Cain, M. K., Zhang, Z., & Yuan, KH. (2024). Univariate and multivariate skewness and kurtosis for measuring nonnormality: Prevalence, influence and estimation. *Behavior Research Methods, 49*, 1716–1735. https://doi.org/10.3758/s13428-016-0814-1

Mardia, K. V. (1970). Measures of multivariate skewness and kurtosis with applications. *Biometrika, 57*(3), 519-530. https://doi.org/10.2307/2334770

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

William Revelle (2024). *psych: Procedures for Psychological, Psychometric, and Personality Research*. Northwestern University, Evanston, Illinois. R package version 2.4.6, https://CRAN.R-project.org/package=psych.

See Also

`descript`

Examples

```
# Example 1a: Compute univariate sample skewness
skewness(mtcars, mpg)

# Example 1b: Compute univariate sample excess kurtosis
kurtosis(mtcars, mpg)

# Example 2a: Compute multivariate skewness
skewness(mtcars)

# Example 2b: Compute multivariate excess kurtosis
kurtosis(mtcars)
```

---

summa *Print Summary Output*

Description

This function prints a summary of the result object returned by the function "lm" for estimating linear regression models and for the result object returned by the function "lmer" from the **lme4** or **lmerTest** package, result object returned by the function "lme" from the **nlme** package, or by the function "rlmer" from the **robustlmm** package to estimate two- or three-level (robust) multilevel and linear mixed-effects models. By default, the function prints the function call, model summary, variance and correlation components, and the regression coefficient table.

Usage

```
summa(model, print = c("all", "default", "call", "descript", "cormat", "modsum",
                       "randeff", "varcor", "coef", "confint", "stdcoef", "vif"),
      robust = FALSE, ddf = c("Satterthwaite", "Kenward-Roger", "lme4"),
      method = c("profile", "wald", "boot"), conf.level = 0.95, nrep = 1000,
      boot = c("perc", "basic", "norm"), seed = NULL, digits = 2, p.digits = 3,
      write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

model	a fitted model of class "lm", "lmerMod", "rlmerMod", "lmerModLmerTest", or "lme"
print	a character vector indicating which results to print, i.e. "all", for all results, "call" for the function call, "descript" for descriptive statistics, cormat for the Pearson product-moment correlation matrix for models estimated by "lm" (see cor.matrix function) or within- and between-group correlation matrix for models estimated by "lmer" or "lme" (see multilevel.cor function), modsum for the multiple correlation, r-squared, and F-test for models estimated by "lm" or model summary, marginal, and conditional R-squared for models estimated by "lmer" or "lme" (see multilevel.r2 function), "randeff" for the random effects (variance and correlation components) for models estimated by "lmer"

or "lme", "varcor" for the variance and correlation structure for models estimated by "lme", coef for the unstandardized coefficients for models estimated by "lm" and fixed effects for models estimated by "lmer" or "lme", confint for the confidence interval for unstandardized coefficients, stdcoef for the standardized coefficients (see coeff.std function), and vif for the variance inflation factor (see check.collin function). The default setting is print = c("call", "modsum", "randeff", "varcor", "coef"). Note that when a fitted model of class "rlmerMod" is specified for the argument model, the default setting is c("call", "randeff", "coef") and that "descript", "cormat", "modsum", "confint", "stdcoef", and "vif" are not available for an "rlmerMod" object.

robust — logical: if TRUE, heteroscedasticity-consistent standard errors, confidence intervals, and heteroscedasticity-robust F-test using the HC4 estimator are computed for linear models estimated by using the lm() function or cluster-robust standard errors using the CR2 estimator is computed for multilevel and linear mixed-effects models estimated by using the lmer() or lme() function (see coeff.robust function).

ddf — a character string for specifying the method for computing the degrees of freedom when using the **lmerTest** package to obtain *p*-values for fixed effects in multilevel and linear mixed-effects models, i.e., "Satterthwaite" (default) for Satterthwaite's method, "Kenward-Roger" for the Kenward-Roger's method, and "lme4" for the lme4-summary without degrees of freedom and significance values (see Kuznetsova et al., 2017). Note that when a fitted model of class "rlmerMod" is specified for the argument model, Satterthwaite or Kenward-Roger degrees of freedom are computed only if the R package **lmerTest** is loaded.

method — a character string for specifying the method for computing confidence intervals (CI), i.e., "profile" (default for "lmer" objects) for computing a likelihood profile and finding the appropriate cutoffs based on the likelihood ratio test, "wald" (default for "lme" objects) for approximating the CIs based on the estimated local curvature of the likelihood surface, and "boot" for performing bootstrapping with CIs computed from the bootstrap distribution according to the argument boot..

conf.level — a numeric value between 0 and 1 indicating the confidence level of the interval.

nrep — a numeric value indicating the number of bootstrap replicates (default is 1000).

boot — a character string for specifying the type of bootstrap confidence intervals (CI), i.e., i.e., "perc" (default), for the percentile bootstrap CI, "basic" for the basic bootstrap CI, and "norm" for the normal approximation bootstrap CI.

seed — a numeric value specifying seeds of the pseudo-random numbers used in the bootstrap algorithm when conducting bootstrapping.

digits — an integer value indicating the number of decimal places to be used.

p.digits — an integer value indicating the number of decimal places to be used for displaying multiple R, R-squared and *p*-value.

write — a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.

`append`	logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.
`check`	logical: if `TRUE` (default), argument specification is checked.
`output`	logical: if `TRUE` (default), output is shown on the console.

Details

Robust Estimation of Multilevel and Linear Mixed-Effects Models The function `rlmer` from the **robustlmm** package does not provide any degrees of freedom or significance values. This function re-estimates the model without using robust estimation to obtain the Satterthwaite or Kenward-Roger degrees of freedom depending on the argument `ddf` before computing significance values for the regression coefficients based on parameter estimates and standard error of the robust multilevel mixed-effects (see Sleegers et al. (2021).

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`args`	specification of function arguments
`result`	list with results, i.e., `call` for the the function call, `descript` for descriptive statistics, `cormat` for the correlation matrix, `modsum` for the model summary, `varcor` for the variance and correlation structure, `randeff` for the variance and correlation components,`coef` for the model coefficients, `weights` for the robustness weights and `converg` for the convergence check, i.e., `1` = model converged, `0` = model singular, and `-1` model not converged.

Author(s)

Takuya Yanagida

References

Kuznetsova, A, Brockhoff, P. B., & Christensen, R. H. B. (2017). lmerTest Package: Tests in linear mixed effects models. *Journal of Statistical Software, 82* 13, 1-26. https://doi.org/10.18637/jss.v082.i13

Sleegers, W. W. A., Proulx, T., & van Beest, I. (2021). Pupillometry and hindsight bias: Physiological arousal predicts compensatory behavior. *Social Psychological and Personality Science, 12*(7), 1146–1154. https://doi.org/10.1177/1948550620966153

See Also

`descript`, `cor.matrix`, `coeff.std`, `coeff.robust`, `check.collin`

Examples

```
#————————————————————————————————————————————————————————————————————
# Linear Model

# Estimate linear model
mod.lm <- lm(mpg ~ cyl + disp, data = mtcars)

# Example 1a: Default setting
summa(mod.lm)

# Example 1b: Heteroscedasticity-consistent standard errors
summa(mod.lm, robust = TRUE)

# Example 1c: Print all available results
summa(mod.lm, print = "all")

# Example 1d: Print default results plus standardized coefficient
summa(mod.lm, print = c("default", "stdcoef"))

## Not run:

#————————————————————————————————————————————————————————————————————
# Multilevel and Linear Mixed-Effects Model

# Load lme4, nlme, and misty package
misty::libraries(lme4, nlme, misty)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#···................
## Two-Level Data

# Cluster-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, x2, type = "CWC", cluster = "cluster")

# Grand-mean centering, center() from the misty package
Demo.twolevel <- center(Demo.twolevel, w1, type = "CGM", cluster = "cluster")

# Estimate two-level mixed-effects model using the lme4 package
mod.lmer2 <- lmer(y1 ~ x2.c + w1.c + x2.c:w1.c + (1 + x2.c | cluster), data = Demo.twolevel)

# Estimate two-level mixed-effects model using the nlme package
mod.lme2 <- lme(y1 ~ x2.c + w1.c + x2.c:w1.c, random = ~ 1 + x2.c | cluster, data = Demo.twolevel)

# Example 2a: Default setting
summa(mod.lmer2)
summa(mod.lme2)

# Example 2b: Print all available results
summa(mod.lmer2, print = "all")
summa(mod.lme2, print = "all")
```

```
# Example 2c: Print default results plus standardized coefficient
summa(mod.lmer2, print = c("default", "stdcoef"))
summa(mod.lme2, print = c("default", "stdcoef"))

# Load lmerTest package
library(lmerTest)

# Re-estimate two-level model using the lme4 and lmerTest package
mod.lmer2 <- lmer(y1 ~ x2.c + w1.c + x2.c:w1.c + (1 + x2.c | cluster), data = Demo.twolevel)

# Example 2d: Default setting, Satterthwaite's method
summa(mod.lmer2)

# Example 2e: Kenward-Roger's method
summa(mod.lmer2, ddf = "Kenward-Roger")

# Example 2f: Cluster-robust standard errors
summa(mod.lmer2, robust = TRUE)

#...................
## Robust Estimation using the R package robustlmm

# Estimate two-level mixed-effects model
mod.lmer2r <- robustlmm::rlmer(y1 ~ x2.c + w1.c + (1| cluster), data = Demo.twolevel)

# Example 2f: Default setting
summa(mod.lmer2r)

#...................
## Three-Level Data

# Create arbitrary three-level data
Demo.threelevel <- data.frame(Demo.twolevel, cluster2 = Demo.twolevel$cluster,
                                             cluster3 = rep(1:10, each = 250))

# Cluster-mean centering, center() from the misty package
Demo.threelevel <- center(Demo.threelevel, x1, type = "CWC", cluster = c("cluster3", "cluster2"))

# Cluster-mean centering, center() from the misty package
Demo.threelevel <- center(Demo.threelevel, w1, type = "CWC", cluster = c("cluster3", "cluster2"))

# Estimate three-level model using the lme4 package
mod.lmer3 <- lmer(y1 ~ x1.c + w1.c + (1 | cluster3/cluster2), data = Demo.threelevel)

# Estimate three-level model using the nlme package
mod.lme3 <- lme(y1 ~ x1.c + w1.c, random = ~ 1 | cluster3/cluster2, data = Demo.threelevel)

# Example 3a: Default setting
summa(mod.lmer3)
summa(mod.lme3)

# Example 3b: Print all available results
```

```
summa(mod.lmer3, print = "all")
summa(mod.lme3, print = "all")

#...................
## Robust Estimation using the R package robustlmm

# Estimate three-level model using the lme4 package
mod.lmer3r <- robustlmm::rlmer(y1 ~ x1.c + w1.c + (1 | cluster3/cluster2),
                               data = Demo.threelevel)

# Example 3c: Default setting
summa(mod.lmer3r)

#=====================================================================
# Write Results

# Example 4a: Write Results into a text file
summa(mod.lm, print = "all", write = "Linear_Model.txt")

# Example 4b: Write Results into a Excel file
summa(mod.lm, print = "all", write = "Linear_Model.xlsx")

## End(Not run)
```

---

test.bm *Brunner-Munzel Test for Stochastic Equality*

Description

This function performs two-sample and paired-sample (Permuted) Brunner-Munzel test (Brunner & Munzel, 2000) using either a t-distribution with a Satterthwaite-Welch degrees of freedom approximation, an exact permutation test, or a randomization test (Neubert & Brunner, 2007).

Usage

```
test.bm(...)

## Default S3 method:
test.bm(data = NULL, ..., x = NULL, y = NULL, paired = FALSE,
      method = c("t", "logit", "perm"), alternative = c("two.sided", "less", "greater"),
       scale = c("prob", "diff", "logodds", "odds"), hypo = FALSE, descript = TRUE,
         conf.level = 0.95, nperm = 10000, seed = 42, digits = 3, p.digits = 3,
         as.na = NULL, write = NULL, append = TRUE, check = TRUE, output = TRUE)

## S3 method for class 'formula'
test.bm(formula, data, method = c("t", "logit", "perm"),
         alternative = c("two.sided", "less", "greater"),
       scale = c("prob", "diff", "logodds", "odds"), hypo = FALSE, descript = TRUE,
       conf.level = 0.95, p.adj = "none", nperm = 10000, seed = 42, na.omit = FALSE,
```

```
digits = 3, p.digits = 3, as.na = NULL, write = NULL, append = TRUE,
check = TRUE, output = TRUE, ...)
```

Arguments

`data`	a data frame. Note that data are specified either by using the argument `data` or by using the arguments `x` and `y`.
`...`	an expression indicating the variable names in `data`, e.g., `test.bm(dat, x1, x2)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`x`	a numeric vector of data values.
`y`	a numeric vector of data values.
`formula`	in case of two sample Brunner-Munzel test (i.e., `paired = FALSE`), a formula of the form `y ~ group` where `group` is a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
`paired`	logical: if `TRUE`, paired-sample Brunner-Munzel-test is computed.
`method`	a character string indicating the method used for statistical significance testing, i.e., `"t"` for a t-distribution with Satterthwaite-Welch degrees of freedom approximation, `"logit"` for a t-distribution with Satterthwaite-Welch degrees of freedom approximation and logit transformation for range-preserving confidence intervals, and `"perm"` for a studentized permutation or randomization test that is recommended for a better control of the type I error rate when sample sizes are small (smaller than 15 per group).
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`scale`	a character string indicating the scale of the reported estimated and confidence interval, i.e., `"prob"` (default) for $P(X < Y)+0.5(P(X = Y)$ with range $[0, 1]$ and a null value of 0.5, `"diff"` for $P(X < Y) - P(X > Y)$ with range $[-1, 1]$ and a null value of 0, `"logodds"` for $log[P/(1 - P)]$ with range $[-Inf, Inf]$ and a null value of 0, and `"odds"` for $P/(1 - P)$ with range $[0, Inf]$ and a null value of 1. Note that the statistical test always operates on the probability scale.
`hypo`	logical: if `TRUE` (default), null and alternative hypothesis are shown on the console.
`descript`	logical: if `TRUE` (default), descriptive statistics are shown on the console.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`p.adj`	a character string indicating an adjustment method for multiple testing based on `p.adjust`, i.e., `none` (default), `bonferroni`, `holm`, `hochberg`, `hommel`, `BH`, `BY`, or `fdr`. See help page of the `p.adjust` function.
`nperm`	a numeric value (default: 10000) indicating the maximum number of permutations used when `method = "perm"`. If the number of complete permutations, i.e., `choose(N, n.x)` in the two-sample design and `2^n` in the paired-sample design, is smaller than or equal `nperm`, an exact permutation test is conducted, while a randomization test based on `nperm` samples of unique permutations is conducted when the number of complete permutations is greater than `nperm`.

seed	a numeric value specifying the seed of the pseudo-random numbers used when drawing permutation samples.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal .places to be used for displaying descriptive statistics and confidence interval.
p.digits	an integer value indicating the number of decimal .places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
sample	type of sample, i.e., two- or paired sample
formula	formula
data	data frame with the outcome and grouping variable
args	specification of function arguments
result	result table

Note

This function is based on the brunner_munzel() and ses_calc() function in the **TOSTER** package by Daniel Lakens and Aaron Caldwell.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Schüürhuis, S., Konietschke, F., & Brunner, E. (2025). A New Approach to the nonparametric Behrens-Fisher problem With compatible confidence iIntervals. *Biometrical Journal, 67*(6), e70096. https://doi.org/10.1002/bimj.70096

Karch, J. D. (2021). Psychologists should use Brunner-Munzel's instead of Mann-Whitney's U Test as the default nonparametric procedure. *Advances in Methods and Practices in Psychological Science, 4*(2). https://doi.org/10.1177/2515245921999

Neubert, K., & Brunner, E. (2007). A studentized permutation test for the non-parametric Behrens-Fisher problem. *Computational Statistics & Data Analysis, 51*(10), 5192–5204. https://doi.org/10.1016/j.csda.2006.05.024

See Also

test.t, test.welch, test.z

Examples

```
#————————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 1a: Two-sided two-sample Brunner-Munzel test
test.bm(mpg ~ vs, data = mtcars)

# Example 1b: Alternative specification without using the '...' argument
test.bm(x = df.subset(mtcars, mpg, subset = vs == 0),
        y = df.subset(mtcars, mpg, subset = vs == 1))

# Example 1c: Two-sided two-sample Brunner-Munzel randomization test
# Sample of 10000 out of choose(32, 18) = 471435600 permutations
test.bm(mpg ~ vs, data = mtcars, method = "perm")

# Example 1d: Two-sided two-sample Brunner-Munzel permutation test
# Complete permutation of choose(13, 8) = 1287 permutations
test.bm(x = c(3, 1, 4, 2, 5, 3, 6, 7), y = c(5, 2, 4, 3, 1), method = "perm")

# Example 1e: One-sided two-sample Brunner-Munzel test
test.bm(mpg ~ vs, data = mtcars, alternative = "greater")

# Example 1f: Multiple two-sample Brunner-Munzel tests
test.bm(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars)

# Example 1g: Multiple two-sample Brunner-Munzel tests, bonferroni correction
test.bm(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars, p.adj = "bonferroni")

#————————————————————————————————————————————————————————————————————
# Paired-Sample Design

# Example 2a: Two-sided paired-sample Brunner-Munzel test
test.bm(mtcars, drat, wt, paired = TRUE)

# Example 2b: Alternative specification without using the '...' argument
test.bm(mtcars[, c("drat", "wt")], paired = TRUE)

# Example 2c: Alternative specification without using the '...' argument
test.bm(x = mtcars$drat, y = mtcars$wt, paired = TRUE)

# Example 2d: Two-sided paired-sample Brunner-Munzel test with logit transformation
test.bm(mtcars, drat, wt, paired = TRUE, method = "logit")

## Not run:
#————————————————————————————————————————————————————————————————————
```

```
# Write Results

# Example 3a: Write results into a text file
test.bm(mpg ~ vs, data = mtcars, write = "BM-Test.txt")

# Example 3b:  Write results into an Excel file
test.bm(mpg ~ vs, data = mtcars, write = "BM-Test.xlsx")

## End(Not run)
```

`test.homo` *Tests for Homoscedasticity in a Linear Model*

Description

This function performs the Breusch-Pagan test (Breusch & Pagan, 1979) without or with the studentising modification of Koenker (1981), the White's test with squared terms or squared and interaction terms (White, 1980), Evans-King Generalized Least Squares (GLS) test (Evans & King, 1988), and the Verbyla's Residual Maximum Likelihood test (Verbyla, 1993) for testing homoskedasticity (null hypothesis) against heteroskedasticity (alternative hypothesis) in a linear regression model. By default, the function performs the Evans-King GLS test.

Usage

```
test.homo(model, method = c("bp", "sbp", "white1", "white2", "evansking", "verbyla"),
          pred = attr(terms(model), which = "term.labels"), lambda.star = 5,
          digits = 2, p.digits = 3,
          plot = FALSE, geom = c("all", "none", "hline", "loess", "ci.loess"),
          filename = NULL, width = NA, height = NA, dpi = 600,
          write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`model`	a fitted model of class `lm`.
`method`	a character string indicating the method used for testing homoscedasticity, i.e., `"bp"` for the the Breusch-Pagan test, `"sbp"` for the Koenker's Studentized Breusch-Pagan test, `"white1"` for the White's Test regressing squared residuals on predictors and their squares, `"white2"` for the White's Test regressing squared residuals on predictors, their squares, and two-way interactions, `"evansking"` (default) for the Evans-King Generalized Least Squares test, and `"verbyla"` for the Verbyla's Residual Maximum Likelihood test.
`pred`	a character string or character vector indicating the predictor variables for the auxiliary design matrix that are suspected to be related to the residual variance under the alternative hypothesis. By default, the design matrix of the regression model including all predictor variables is used.

lambda.star	a numeric value (default: 5) indicating the degree of heteroskedasticity under the alternative hypothesis for Evans-King Generalized Least Squares test (method = "evansking"). Evans and King (1985) suggests 2.5, 5, 7.5, and 10 as values to consider, and Evans and King (1988) finds that 2.5 and 5 perform best empirically.
plot	logical: if TRUE, a plot for predicted values against the studentized residuals is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.
geom	a character vector indicating which plot elements to draw, i.e., "all" (default) for drawing all plot elements, "none" for drawing no horizontal or smoothing line, "hline" for drawing a horizontal line at 0, "loess" for drawing the LOESS smoothing curve, and "ci.loess" for drawing confidence bands around the smoothing curve (i.e., se = TRUE).
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
digits	an integer value indicating the number of decimal places to be used for displaying the test statistic.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

Farrar et al. (2025) reviewed 25 different tests for homoscedasticity implemented in the R package **scedastic** (Farrar, 2025) and presented results of a Monte Carlo simulation demonstrating that the Evans-King GLS test and Verbyla's Residual Maximum Likelihood test outperform commonly used tests for homoskedasticity, such as the Koenker's Studentized Breusch-Pagan test, White's test, and the Goldfeld-Quandt F test.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`model`	model specified in `model`
`plot`	ggplot2 object for plotting the results
`args`	specification of function arguments
`result`	data frame with results

Note

This function is based on the `breusch_pagan()`, `white()`, `evans_king()`, `pRQF()` and `verbyla()` function in the **skedastic** package by Thomas Farrar.

Author(s)

Takuya Yanagida

References

Breusch, T. S., & Pagan, A.R. (1979) A simple test for heteroscedasticity and Random Coefficient Variation. *Econometrica, 47*, 1287-1294. http://dx.doi.org/10.2307/1911963

Evans, M. A., & King, M. L. (1985). A point optimal test for heteroscedastic disturbances. *Journal of Econometrics, 27*(2), 163–178.

Evans, M. A., & King, M. L. (1988). A further class of tests for heteroscedasticity. *Journal of Econometrics, 37*(2), 265–276.

Farrar, T. J. (2025). *skedastic: Heteroskedasticity Diagnostics for Linear Regression Models*. R Package version 2.0.3. https://github.com/tjfarrar/skedastic

Farrar, T., Blignaut, R., Luus, R., & Steel, S. (2025). A review and comparison of methods of testing for heteroskedasticity in the linear regression model. *Journal of applied statistics, 52*(16), 3121-3150. https://doi.org/10.1080/02664763.2025.2575038

Koenker, R. (1981). A note on studentizing a test for heteroscedasticity. *Journal of Econometrics, 17*(1), 107–112. https://doi.org/10.1016/0304-4076(81)90062-2.

Verbyla, A. P. (1993). Modelling variance heterogeneity: Residual maximum likelihood and diagnostics. *Journal of the Royal Statistical Society: Series B (Methodological), 55*(2), 493–508.

White, H. (1980). A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica, 48*(4), 817–838. https://doi.org/10.2307/1912934

See Also

`check.resid`, `test.linear`, `test.norm`, `coeff.robust`

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————
# Tests for Homoscedasticity in a Linear Model

# Estimate linear model
mod <- lm(mpg ~ cyl + disp, data = mtcars)

# Example 1a: Koenker's Studentized Breusch-Pagan test (default)
test.homo(mod)

# Example 1b: White's Test regressing squared residuals on predictors and their squares
test.homo(mod, method = "white1")

# Example 1c: Evans-King Generalized Least Squares test
test.homo(mod, method = "evansking")

# Example 1d: Verbyla's Residual Maximum Likelihood test
test.homo(mod, method = "verbyla")

# Example 2: Predictor variable 'disp' used in the auxiliary design matrix
test.homo(mod, pred = "disp")

#————————————————————————————————————————————————————————————————————
# Plot

# Example 3a: Predicted values vs. Studentized residuals
test.homo(mod, plot = TRUE)

# Example 3b: Plot with horizontal line only
test.homo(mod, plot = TRUE, geom = "hline")

# Example 3c: redicted values vs. Studentized residuals, additional arguments
# see Details in the help page of the function plot.misty.object
test.homo(mod, plot = TRUE, point.shape = 2, linetype = "solid", line.col = "blue",
          xlim = c(0, 30))

# Example 3d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.homo(mod)
plot(object, point.shape = 2, linetype = "solid", line.col = "blue", xlim = c(0, 30))

#————————————————————————————————————————————————————————————————————
# Write Results

# Example 4a: Write results into a text file
test.homo(mod, write = "Test_Homo.txt")

# Example 4b: Write results into an Excel file
test.homo(mod, write = "Test_Homo.xlsx")

# Example 4c: Save plot as PNG fine
```

```
test.homo(mod, plot = TRUE, filename = "Homo.png", width = 7, height = 6)

## End(Not run)
```

`test.levene` *Levene's Test for Homogeneity of Variance*

Description

This function performs Levene's test for homogeneity of variance across two or more independent groups including a plot showing violin plots and boxplots representing the distribution of the outcome variable for each group.

Usage

```
test.levene(formula, data, method = c("median", "mean"), hypo = FALSE,
            descript = FALSE, conf.level = 0.95, digits = 2, p.digits = 3,
          as.na = NULL, plot = FALSE, geom = c("all", "violin", "box", "jitter"),
            gray = FALSE, filename = NULL, width = NA, height = NA, dpi = 600,
            write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`formula`	a formula of the form `y ~ group` where `y` is a numeric variable giving the data values and `group` a numeric variable, character variable or factor with two or more than two values or factor levels giving the corresponding groups.
`data`	a matrix or data frame containing the variables in the formula `formula`.
`method`	a character string specifying the method to compute the center of each group, i.e. `method = "median"` (default) to compute the Levene's test based on the median (aka Brown-Forsythe test) or `method = "mean"` to compute the Levene's test based on the arithmetic mean.
`hypo`	logical: if `TRUE`, null and alternative hypothesis are shown on the console.
`descript`	logical: if `TRUE`, descriptive statistics are shown on the console.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`plot`	logical: if `TRUE`, a plot showing violins with boxplots is drawn.
`geom`	a character string or character vector indicating which plot elements to draw, i.e., `"all"` for drawing all plot elements, `"violin"` for drawing violins, `"box"` for drawing boxplots, and `"jitter"` for drawing jittered data points.

gray	logical: if TRUE, the plot is drawn in gray scale.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when specifying plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default: 600) for the ggsave function.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Details

Levene's Test The Levene's test is equivalent to a one-way analysis of variance (ANOVA) with the absolute deviations of observations from the mean of each group as dependent variable (center = "mean"). Brown and Forsythe (1974) modified the Levene's test by using the absolute deviations of observations from the median (center = "median"). By default, the Levene's test uses the absolute deviations of observations from the median.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	data frame with the outcome and grouping variable
formula	formula
args	specification of function arguments
plot	ggplot2 object for plotting the results
result	result table

Author(s)

Takuya Yanagida

References

Brown, M. B., & Forsythe, A. B. (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association, 69*, 364-367.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

aov.b, test.t, test.welch

Examples

```
#————————————————————————————————————————————————————————————————————
# Levene's Test

# Example 1a: Levene's test based on the median
test.levene(mpg ~ gear, data = mtcars)

# Example 1b: Levene's test based on the arithmetic mean
test.levene(mpg ~ gear, data = mtcars, method = "mean")

# Example 1c: Levene's test, print descriptive statistics
test.levene(mpg ~ gear, data = mtcars, descript = TRUE)

#————————————————————————————————————————————————————————————————————
# Plot

# Example 2a: Plot results, default setting
test.levene(mpg ~ gear, data = mtcars, plot = TRUE)

# Example 2b: Plot results, no violin plots, draw jittered data points
test.levene(mpg ~ gear, data = mtcars, plot = TRUE, geom = "jitter")

# Example 2c: Plot results, additional arguments
# see Details in the help page of the function plot.misty.object
test.levene(mpg ~ gear, data = mtcars, plot = TRUE, violin.alpha = 0.1,
            box.width = 0.1, title = "Levene's Test")

# Example 2d: Plot results using the plot() function, additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.levene(mpg ~ gear, data = mtcars)
plot(object, violin.alpha = 0.1, box.width = 0.1, title = "Levene's Test")

#————————————————————————————————————————————————————————————————————
# Create Plot Manually

# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- test.levene(mpg ~ gear, data = mtcars)
```

```
# Example 3: Plot
ggplot(object$data, aes(group, y, fill = group)) +
  geom_violin(alpha = 0.3, trim = FALSE) +
  geom_boxplot(alpha = 0.2, width = 0.2) +
  geom_jitter(alpha = 0.2, width = 0.05, height = 0, size = 1.25) +
  theme_bw() +
  guides(fill = "none")

## Not run:
#————————————————————————————————————————————————————————————————————————
# Write Results and Save Plot

# Example 4a: Write results into a text file
test.levene(mpg ~ gear, data = mtcars, write = "Levene.txt")

# Example 4b: Write results into an Excel file
test.levene(mpg ~ gear, data = mtcars, write = "Levene.xlsx")

# Example 4c: Save plot as PNG fine
test.levene(mpg ~ gear, data = mtcars, plot = TRUE,
            filename = "Levene-Test.png", width = 6, height = 5)

## End(Not run)
```

`test.linear` *Tests for Linearity in a Linear Model*

Description

This function performs the regression equation specification error (RESET) test (Remsey, 1969), the Harvey-Collier test (Harvey & Collier, 1977), and the rainbow test for linearity (Utts, 1982). By default, the function performs the RESET test.

Usage

```
test.linear(model, method = c("reset", "harv", "rain"), power = c(2, 3),
            frac = 0.5, order = c("none", "all", "mahalanobis"), p.adj = "none",
             digits = 2, p.digits = 3, plot = FALSE,
             geom = c("all", "none", "lm", "loess", "ci.lm", "ci.loess"),
             filename = NULL, width = NA, height = NA, dpi = 600,
             write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`model`	a fitted model of class `lm`.
`method`	a character string indicating the method used for testing linearity, i.e., `"reset"` (default) for the regression equation specification error (RESET) test, `"harv"` for the Harvey-Collier test, and `"rain"` for the rainbow test for linearity.

power — a positive integer vector indicating the powers of the fitted values that should be included in the auxiliary regression model. By default, the test includes quadratic and cubic terms in the model, i.e., `c(2, 3)`. Note that this argument is only used for the RESET test.

frac — a numeric value between 0 and 1 indicating the proportion of observations used in the central subset model. Note that this argument is only used for the rainbow test.

order — a character string indicated how the observations are ordered, i.e., `"none"` for observations assumed to be already ordered, `"all"` for ordering each predictor in the model specified in the `"model"` argument separately, `"mahalanobis"` for ordering observations by their Mahalanobis distances from the mean regressor. Note that this argument is only used for the Harvey-Collier test and the rainbow test.

p.adj — a character string indicating an adjustment method for multiple testing based on `p.adjust`, i.e., `none` (default), `bonferroni`, `holm`, `hochberg`, `hommel`, `BH`, `BY`, or `fdr`.

digits — an integer value indicating the number of decimal places to be used for displaying results.

p.digits — an integer value indicating the number of decimal places to be used for displaying the *p*-value.

plot — logical: if `TRUE`, partial residual plots or component-plus-residual plots are drawn. For additional plotting arguments, see `Details` in the help page of the function `plot.misty.object`.

geom — character vector indicating which plot elements to draw, i.e., `"all"` (default) for drawing all plot elements, `"none"` for drawing no regression or smoothing line, `"lm"` for drawing the regression line, `"loess"` for drawing the LOESS smoothing curve, `"ci.lm"` for drawing confidence bands around the regression line, `"ci.loess"` for drawing confidence bands around the LOESS smoothing curve (i.e., `se = TRUE`).

filename — a character string indicating the `filename` argument including the file extension in the `ggsave` function. Note that one of `".eps"`, `".ps"`, `".tex"`, `".pdf"` (default), `".jpeg"`, `".tiff"`, `".png"`, `".bmp"`, `".svg"` or `".wmf"` needs to be specified as file extension in the `file` argument. Note that plots can only be saved when `plot = TRUE`.

width — a numeric value indicating the `width` argument (default is the size of the current graphics device) for the `ggsave` function.

height — a numeric value indicating the `height` argument (default is the size of the current graphics device) for the `ggsave` function.

dpi — a numeric value indicating the `dpi` argument (default is `600`) for the `ggsave` function.

write — a character string naming a text file with file extension `".txt"` (e.g., `"Output.txt"`) for writing the output into a text file.

append — logical: if `TRUE` (default), output will be appended to an existing text file with extension `.txt` specified in `write`, if `FALSE` existing text file will be overwritten.

check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown.
...	additional plotting arguments from the plot.misty.object() function, see 'Details in the plot.misty.object function.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
model	model specified in model
plot	ggplot2 object for plotting the results
args	specification of function arguments
result	data frame with results

Note

This function is based on the resettest(), harvtest(), and raintest() function in the **lmtest** package by Achim Zeileis and Torsten Hothorn.

Author(s)

Takuya Yanagida

References

Harvey, A., & Collier, P. (1977). Testing for functional misspecification in regression analysis. *Journal of Econometrics, 6*(1), 103-119.

Ramsey, J. B. (1969). Tests for specification errors in classical linear least-squares regression analysis. *Journal of the Royal Statistical Society: Series B (Methodological), 31*(2), 350-371.

Utts, J. M. (1982). The rainbow test for lack of fit in regression. *Communications in Statistics - Theory and Methods, 11*(24), 2801-2815.

Zeileis, A., & Hothorn, T. (2002). Diagnostic checking in regression relationships. *R News, 2*(3), 7-10.

See Also

check.resid, test.homo, test.norm, coeff.robust

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————————
# Tests for Linearity in a Linear Model

# Estimate linear model
mod <- lm(mpg ~ disp + hp + drat, data = mtcars)
```

```
# Example 1a: RESET test (default)
test.linear(mod)

# Example 1b: Harvey-Collier test
test.linear(mod, method = "harv")

# Example 1c: Rainbow test
test.linear(mod, method = "rain")

#————————————————————————————————————————————————————————————————————————
# 'order' Argument

# Example 2a: Harvey-Collier test, order observations by 'hp'
test.linear(mod, method = "harv", order = "hp")

# Example 2a: Rainbow test, order by Mahalanobis distances
test.linear(mod, method = "rain", order = "mahalanobis")

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 3a: Partial residual plots
test.linear(mod, plot = TRUE)

# Example 3b: Plot with smoothing line only
test.linear(mod, plot = TRUE, geom = "loess")

# Example 3c: Partial residual plots, additional arguments
# see Details in the help page of the function plot.misty.object
test.linear(mod, plot = TRUE, point.size = 1, point.shape = 2)

# Example 3d: Plot results using the plot() function, additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.linear(mod)
plot(object, point.size = 1, point.shape = 2)

#————————————————————————————————————————————————————————————————————————
# Write Results

# Example 4a: Write results into a text file
test.linear(mod, write = "Test_Linear.txt")

# Example 4b: Write results into an Excel file
test.linear(mod, write = "Test_Linear.xlsx")

# Example 4c: Save plot as PNG fine
test.linear(mod, plot = TRUE, filename = "Linear.png", width = 15, height = 5)

## End(Not run)
```

test.norm *Tests for Univariate and Multivariate Normality*

Description

This function performs the Kolmogorov-Smirnov-Lilliefors, the Shapiro-Wilk test, the Anderson–Darling test, and the Cramer-von Mises test for testing for testing univariate normality (null hypothesis) against univariate non-normality (alternative hypothesis), and the Henze-Zirkler test, Henze-Wagner test, and the Royston test for testing multivariate normality (null hypothesis) against multivariate non-normality (alternative hypothesis). By default, the function performs the Shapiro-Wilk test for testing univariate normality, and Henze-Zirkler test for testing multivariate normality. Note that the function also tests normality of residuals of a linear model when specifying an object of class "lm" for the argument data.

Usage

```
test.norm(data, ..., print = c("all", "multi", "uni"),
          method.uni = c("lillie", "sw", "ad", "cm"),
          method.multi = c("hz", "hw", "royston"), p.adj = "none", digits = 2,
         p.digits = 3, as.na = NULL, plot = FALSE, geom = c("all", "qq", "hist"),
          impute = FALSE, filename = NULL, width = NA, height = NA, dpi = 600,
          write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

data	a fitted model of class lm, a numeric vector, or data frame.
...	an expression indicating the variable names in data, e.g., test.norm(dat, x1, x2). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
print	a character vector indicating which results to show on the console, i.e. "all" for tests for multivariate and univariate normality, "multi" (default) for a test for multivariate normality, and "uni" for tests for univariate normality. Note that this argument is only used when specifying more than one variable for the analysis.
method.uni	a character string indicating the method used for testing univariate normality, i.e., "lillie" for the Kolmogorov-Smirnov-Lilliefors test (Lilliefors, 1967), "sw" (default) for the Shapiro-Wilk test (Shapiro & Wilk, 1965), "ad" for the Anderson–Darling test (Anderson & Darling, 1954), and "cm" for the Cramer-von Mises test (Stephens, 1986).
method.multi	a character string indicating the method used for testing multivariate normality, i.e., "hz" (default) for the Henze-Zirkler test (Henze & Zirkler, 1990), "hw" for the Henze-Wagner test (Henze & Wagner, 1997), and "royston" for the Royston test (Royston, 1983).
p.adj	a character string indicating an adjustment method for multiple testing based on p.adjust, i.e., none (default), bonferroni, holm, hochberg, hommel, BH, BY, or fdr.

digits	an integer value indicating the number of decimal places to be used for displaying results.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	logical: if TRUE, a plot showing univariate or multivariate Mahalanobis Q-Q plot is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.
geom	a character vector indicating which plot to draw, i.e., "all" for plotting a Q-Q plot and histogram, "qq" for plotting Q-Q plots, and "hist" for plotting histograms. Note that "all" and c("qq", "hist") are valid only when specifying a fitted model of class lm or a single variable. When specifying "all" or c("qq", "hist") given multiple variables, a multivariate Mahalanobis Q-Q plot is drawn.
impute	logical: if TRUE, missing values are imputed when plotting multiple variables plot = TRUE using the mice function in the R package **mice** using the predictive mean matching method. Note that listwise deletion is always used for tests for univariate and multivariate normality.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
data	numeric vector or data frame

`plot`	ggplot2 object for plotting the results
`args`	specification of function arguments
`result`	list with result tables, i.e., `uni` for test(s) for univariate normality and `multi` for test for multivariate normality

Note

This function is based on the `hz()`, `hw()`, and `royston()` function in the **MVN** package by Selcuk Korkmaz, Dincer Goksuluk and Gokmen Zararsiz, and the `ad.test()`, `lillie.test()`, and `cvm.test()` function in the **nortest** package by Juergen Gross and Uwe Ligges.

Author(s)

Takuya Yanagida

References

Anderson, T. W., & Darling, D. A. (1954). A test of goodness of fit. *Journal of the American Statistical Association, 49*(268), 765-769. https://doi.org/10.1080/01621459.1954.10501232

Gross J, & Ligges U (2015). *nortest: Tests for Normality*. R package version 1.0-4, https://doi.org/10.32614/CRAN.package.n

Henze, N., & Zirkler, B. (1990). A class of invariant consistent tests for multivariate normality. *Communications in Statistics—Theory and Methods, 19*(10), 3595-3617. https://doi.org/10.1080/03610929008830400

Henze, N., & Wagner, T. (1997). A new approach to the BHEP tests for multivariate normality. *Journal of Multivariate Analysis, 62*(1), 1-23

Korkmaz S, Goksuluk D, & Zararsiz G. (2014) MVN: An R Package for Assessing Multivariate Normality. *The R Journal, 6*(2), 151-162.

Lilliefors, H. W. (1967). On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association, 62*(318), 399-402. https://doi.org/10.1080/01621459.1967.104829

Royston, J.P. (1983). Some techniques for assessing multivariate normality based on the Shapiro-Wilk W. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 32*, 121-133. https://doi.org/10.2307/2347291

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika, 52*(3–4), 591–611. https://doi.org/10.1093/biomet/52.3-4.591

Stephens, M. A. (1986). Tests based on EDF statistics. In R. B. D'Agostino & M. A. Stephens (Eds.), *Goodness-of-Fit Techniques* (pp. 97.193). Marcel Dekker.

See Also

`skewness`, `kurtosis`, `check.resid`, `test.linear`, `test.homo`, `test.levene`

Examples

```
## Not run:
#————————————————————————————————————————————————————————————————————
# Fitted Model of Class "lm"

# Estimate linear model
mod <- lm(mpg ~ cyl + disp, data = mtcars)
```

```
# Example 1a: Shapiro-Wilk test for normality of residuals
test.norm(mod)

# Example 1b: Lilliefors (Kolomorov-Smirnov) test for normality of residuals
test.norm(mod, method.uni = "lillie")

#————————————————————————————————————————————————————————————————————————
# One Variable

# Example 2: Shapiro-Wilk test for normality
test.norm(mtcars, mpg)

#————————————————————————————————————————————————————————————————————————
# Multiple Variables

# Example 3a: Henze-Zirkler test for multivariate normality
test.norm(mtcars, hp, drat, wt, qsec)

# Example 3b: Henze-Wagner test for multivariate normality
test.norm(mtcars, hp, drat, wt, qsec, method.multi = "hw")

# Example 3c: Multiple Shapiro-Wilk tests for normality
test.norm(mtcars, hp, drat, wt, qsec, print = "uni")

# Example 3c: Henze-Wagner test for multivariate normality
#             Multiple Shapiro-Wilk tests for normality
test.norm(mtcars, hp, drat, wt, qsec, print = "all")

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 4a: Q-Q plot and histogram
test.norm(mtcars, mpg, plot = TRUE)

# Example 4b: Multivariate Mahalanobis Q-Q plot
test.norm(mtcars, hp, drat, wt, qsec, plot = TRUE)

# Example 4c: Multiple Q-Q plots
test.norm(mtcars, hp, drat, wt, qsec, print = "uni", plot = TRUE)

# Example 4d: Multiple histograms
test.norm(mtcars, hp, drat, wt, qsec, print = "uni", plot = TRUE, geom = "hist")

# Example 4e: Multivariate Mahalanobis Q-Q plot, impute data for plotting
test.norm(airquality, plot = TRUE, impute = TRUE)

# Example 4f: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.norm(mtcars, mpg)
plot(object, point.shape = 2, ribbon.fill = "red", xlim = c(0, 40))

#————————————————————————————————————————————————————————————————————————
```

```
# Write Results and Save Plot

# Example 5a: Write results into a text file
test.norm(mtcars, hp, drat, wt, qsec, write = "Multinorm.txt")

# Example 5b: Write results into an Excel file
test.norm(mtcars, hp, drat, wt, qsec, write = "Multinorm.xlsx")

# Example 5c: Save plot as PNG fine
test.norm(mtcars, mpg, plot = TRUE, filename = "Multinorm.png",
          width = 13, height = 6)

## End(Not run)
```

---

test.t *t-Test*

Description

This function performs one-sample, two-sample, and paired-sample t-tests and provides descriptive statistics, effect size measure, and a plot showing bar plots with error bars for (difference-adjusted) confidence intervals.

Usage

```
test.t(...)

## Default S3 method:
test.t(data = NULL, ..., x = NULL, y = NULL, mu = 0,
       paired = FALSE, alternative = c("two.sided", "less", "greater"),
       hypo = FALSE, descript = TRUE, effsize = FALSE, weighted = TRUE,
      correct = FALSE, conf.level = 0.95, digits = 2, p.digits = 3, as.na = NULL,
       plot = FALSE, geom = c("all", "bar", "point", "ci", "line", "jitter"),
       adjust = TRUE, filename = NULL, width = NA, height = NA, dpi = 600,
       write = NULL, append = TRUE, check = TRUE, output = TRUE)

## S3 method for class 'formula'
test.t(formula, data, alternative = c("two.sided", "less", "greater"),
     hypo = FALSE, descript = TRUE, effsize = FALSE, weighted = TRUE, correct = FALSE,
      conf.level = 0.95, p.adj = "none", na.omit = FALSE, digits = 2, p.digits = 3,
      as.na = NULL, plot = FALSE, geom = c("all", "bar", "point", "ci", "jitter"),
       adjust = TRUE, filename = NULL, width = NA, height = NA, dpi = 600,
       write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

data	a data frame. Note that data are specified either by using the argument data or by using the arguments x and y.

...	an expression indicating the variable names in data, e.g., test.t(dat, x1, x2). Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
x	a numeric vector of data values.
y	a numeric vector of data values.
formula	in case of two sample t-test (i.e., paired = FALSE), a formula of the form y ~ group where group is a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
mu	a numeric value indicating the population mean under the null hypothesis. Note that the argument mu is only used when computing a one sample t-test.
paired	logical: if TRUE, paired-sample t-test is computed.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
hypo	logical: if TRUE (default), null and alternative hypothesis are shown on the console.
descript	logical: if TRUE (default), descriptive statistics are shown on the console.
effsize	logical: if TRUE, effect size measure Cohen's d is shown on the console, see cohens.d function.
weighted	logical: if TRUE (default), the weighted pooled standard deviation is used to compute Cohen's d for a two-sample design (i.e., paired = FALSE), while standard deviation of the difference scores is used to compute Cohen's d for a paired-sample design (i.e., paired = TRUE).
correct	logical: if TRUE, correction factor to remove positive bias in small samples is used when computing Cohen's d.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
p.adj	a character string indicating an adjustment method for multiple testing based on p.adjust, i.e., none (default), bonferroni, holm, hochberg, hommel, BH, BY, or fdr. See help page of the p.adjust function.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal places to be used for displaying descriptive statistics and confidence interval.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	logical: if TRUE, a plot showing bar plots with error bars for confidence intervals is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "bar" for drawing bars representing means for each groups, "point" for drawing points representing means for each group, "ci" for drawing error bars representing confidence intervals, "jitter" for drawing jittered data points, "line" for drawing a horizontal line at mu for the one-sample t-test or at 0 for the paired-sample t-test.

adjust	logical: if TRUE (default), difference-adjustment for the confidence intervals in a two-sample design is applied.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
sample	type of sample, i.e., one-, two-, or paired sample
formula	formula
data	data frame with the outcome and grouping variable
args	specification of function arguments
plot	ggplot2 object for plotting the results
result	result table

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

aov.b, aov.w, test.welch, test.z, test.levene, cohens.d, ci.mean.diff, ci.mean

Examples

```
#————————————————————————————————————————————————————————————————————————
# One-Sample Design

# Example 1a: Two-sided one-sample t-test, population mean = 20
test.t(mtcars, mpg, mu = 20)

# Example 1b: Alternative specification without using the '...' argument
test.t(mtcars$mpg, mu = 20)

# Example 1c: One-sided one-sample t-test, population mean = 20, print Cohen's d
test.t(mtcars, mpg, mu = 20, alternative = "greater", effsize = TRUE)

#————————————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 2a: Two-sided two-sample t-test
test.t(mpg ~ vs, data = mtcars)

# Example 2b: Alternative specification without using the '...' argument
test.t(x = df.subset(mtcars, mpg, subset = vs == 0),
       y = df.subset(mtcars, mpg, subset = vs == 1))

# Example 2c: One-sided two-sample t-test, print Cohen's d
test.t(mpg ~ vs, data = mtcars, alternative = "greater", effsize = TRUE)

# Example 2d: Multiple two-sample t-tests
test.t(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars)

# Example 2e: Multiple two-sample t-tests, bonferroni correction
test.t(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars, p.adj = "bonferroni")

#————————————————————————————————————————————————————————————————————————
# Paired-Sample Design

# Example 3a: Two-sided paired-sample t-test
test.t(mtcars, drat, wt, paired = TRUE)

# Example 3b:  Alternative specification without using the '...' argument
test.t(mtcars[, c("drat", "wt")], paired = TRUE)

# Example 3c: Alternative specification without using the '...' argument
test.t(x = mtcars$drat, y = mtcars$wt, paired = TRUE)

# Example 3d: One-sided paired-sample t-test, print Cohen's d
test.t(mtcars, drat, wt, paired = TRUE, alternative = "greater", effsize = TRUE)

#————————————————————————————————————————————————————————————————————————
# Plot

# Example 4a: One-Sample Design
test.t(mtcars, mpg, mu = 20, plot = TRUE)
```

```
# Example 4b: Two-Sample Design
test.t(mpg ~ vs, data = mtcars, plot = TRUE)

# Example 4c: Paired-Sample Design
test.t(mtcars, drat, wt, paired = TRUE, plot = TRUE)

# Example 4d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.t(mpg ~ vs, data = mtcars)
plot(object, geom = c("bar", "ci", "jitter"), jitter.alpha = 0.4, title = "Two-Sample t-Test")

#──────────────────────────────────────────────────────────────────────────────
# Create Plot Manually

# Load ggplot2 package
library(ggplot2)

# Example 4a: Two-sample t-test
ci.table <- ci.mean(mtcars, mpg, group = "vs", adjust = TRUE, output = FALSE)$result

ggplot(ci.table, aes(group, m)) +
  geom_bar(aes(group, m), stat = "summary", fun = "mean") +
  geom_errorbar(aes(group, m, ymin = low, ymax = upp), width = 0.1) +
  theme_bw()

# Example 4b: Paired-sample t-test
object <- test.t(mtcars, drat, wt, paired = TRUE)

ggplot(data.frame(x = object$data$y - object$data$x), aes(x = 0L, y = x)) +
 geom_bar(data = object$result, aes(0, m.diff), stat = "summary", fun = "mean") +
 geom_errorbar(data = object$result, aes(0, m.diff, ymin = m.low, ymax = m.upp), width = 0.1) +
 geom_hline(yintercept = 0L, linetype = 3, linewidth = 0.8) +
 scale_x_continuous(name = "", limits = c(-2, 2)) +
 theme_bw()  +
 theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())

## Not run:
#──────────────────────────────────────────────────────────────────────────────
# Write Results and Save Plot

# Example 6a: Write results into a text file
test.t(mpg ~ vs, data = mtcars, write = "t-Test.txt")

# Example 6b:  Write results into an Excel file
test.t(mpg ~ vs, data = mtcars, write = "t-Test.xlsx")

# Example 6c: Save plot as PNG fine
test.t(mpg ~ vs, data = mtcars, plot = TRUE, filename = "t-Test.png",
       width = 6, height = 5)

## End(Not run)
```

test.welch *Welch's Test*

Description

This function performs Welch's two-sample t-test and Welch's ANOVA including Games-Howell post hoc test for multiple comparison and provides descriptive statistics, effect size measures, and a plot showing bars representing means for each group and error bars for difference-adjusted confidence intervals.

Usage

```
test.welch(formula, data, alternative = c("two.sided", "less", "greater"),
           hypo = FALSE, descript = c(FALSE, TRUE), effsize = FALSE,
          weighted = FALSE, correct = FALSE, posthoc = FALSE, conf.level = 0.95,
         p.adj = "none", na.omit = FALSE, digits = 2, p.digits = 3, as.na = NULL,
           plot = FALSE, geom = c("all", "bar", "point", "ci", "jitter"),
           adjust = TRUE, filename = NULL, width = NA, height = NA, dpi = 600,
           write = NULL, append = TRUE, check = TRUE, output = TRUE)
```

Arguments

formula	a formula of the form y ~ group where y is a numeric variable giving the data values and group a numeric variable, character variable or factor with two or more than two values or factor levels giving the corresponding groups.
data	a matrix or data frame containing the variables in the formula formula.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". Note that this argument is only used when conducting Welch's two-sample t-test.
hypo	logical: if TRUE (default), null and alternative hypothesis are shown on the console.
descript	logical: if TRUE, descriptive statistics are shown on the console when conducting Welch's ANOVA.
effsize	logical: if TRUE, effect size measure Cohen's d for Welch's two-sample t-test (see cohens.d), η^2 and ω^2 for Welch's ANOVA and Cohen's d for the post hoc tests are shown on the console.
weighted	logical: if TRUE, the weighted pooled standard deviation is used to compute Cohen's d.
correct	logical: if TRUE, correction factor to remove positive bias in small samples for is used to compute Cohen's d.
posthoc	logical: if TRUE, Games-Howell post hoc test for multiple comparison is conducted when performing Welch's ANOVA.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.

p.adj	a character string indicating an adjustment method for multiple testing based on p.adjust, i.e., none (default), bonferroni, holm, hochberg, hommel, BH, BY, or fdr. See help page of the p.adjust function.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal places to be used for displaying results.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	logical: if TRUE, a plot showing bar plots with error bars for confidence intervals is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "bar" for drawing bars representing means for each groups, "point" for drawing points representing means for each group, "ci" for drawing error bars representing confidence intervals, and "jitter" for drawing jittered data points.
adjust	logical: if TRUE (default), difference-adjustment for the confidence intervals is applied.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the filename argument. Note that plots can only be saved when plot = TRUE.
width	a numeric value indicating the width argument (default is the size of the current graphics device) in the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) in the ggsave function.
dpi	a numeric value indicating the dpi argument (default: 600) in the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Effect Size Measure By default, Cohen's d based on the non-weighted standard deviation (i.e., weighted = FALSE) which does not assume homogeneity of variance is computed (see Delacre et al., 2021) when requesting an effect size measure (i.e., effsize = TRUE). Cohen's d based on the pooled standard deviation assuming equality of variances between groups can be requested by specifying weighted = TRUE.

Value

Returns an object of class `misty.object`, which is a list with following entries:

`call`	function call
`type`	type of analysis
`sample`	type of sample, i.e., one-, two-, or paired sample
`data`	data frame with the outcome and grouping variable
`formula`	formula
`args`	specification of function arguments
`plot`	ggplot2 object for plotting the results
`result`	result table

Author(s)

Takuya Yanagida

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Delacre, M., Lakens, D., Ley, C., Liu, L., & Leys, C. (2021). Why Hedges' g*s based on the non-pooled standard deviation should be reported with Welch's t-test. https://doi.org/10.31234/osf.io/tu6mp

See Also

`test.t`, `test.z`, `test.levene`, `aov.b`, `cohens.d`, `ci.mean.diff`, `ci.mean`

Examples

```
#————————————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 1a: Two-sided two-sample Welch-test
test.welch(hp ~ am, data = mtcars)

# Example 1b: One-sided two-sample Welch-test
test.welch(hp ~ am, data = mtcars, alternative = "less")

# Example 1c: Two-sided two-sample Welch-test,
# Print descriptive statistics and Cohen's d
test.welch(hp ~ am, data = mtcars, descript = TRUE, effsize = TRUE)

# Example 1d: Multiple two-sample Welch-test
test.welch(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars)

# Example 1e: Multiple two-sample Welch-test, bonferroni correction
test.welch(cbind(mpg, cyl, disp, hp, drat, wt) ~ vs, data = mtcars, p.adj = "bonferroni")
```

```
#──────────────────────────────────────────────────────────────────────────────
# Multiple-Sample Design

# Example 2a: Welch's ANOVA
test.welch(hp ~ gear, data = mtcars)

# Example 2b: Welch's ANOVA,
# Print descriptive statistics and Games-Howell post hoc test
test.welch(hp ~ gear, data = mtcars, descript = TRUE, posthoc = TRUE)

# Example 2c: Welch's ANOVA, print eta-squared and omega-squared
test.welch(hp ~ gear, data = mtcars, effsize = TRUE)

#──────────────────────────────────────────────────────────────────────────────
# Plot

# Example 3a: Plot results, default setting
test.welch(hp ~ gear, data = mtcars, plot = TRUE)

# Example 3b: Plot results
# No bars, draw points representing means and jittered data points
test.welch(hp ~ gear, data = mtcars, plot = TRUE, geom = c("point", "jitter"))

# Example 3c: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.welch(hp ~ gear, data = mtcars)
plot(object, geom = c("bar", "ci", "jitter"), jitter.alpha = 0.4, title = "Welch's Test")

#──────────────────────────────────────────────────────────────────────────────
# Create Plot Manually

# Load ggplot2 package
library(ggplot2)

# Create misty object
object <- test.welch(hp ~ gear, data = mtcars)

# Example 4: Plot
ggplot(object$result$descript, aes(group, y)) +
  geom_bar(aes(group, m), stat = "summary", fun = "mean") +
  geom_errorbar(aes(group, m, ymin = low, ymax = upp), width = 0.1) +
  theme_bw()


## Not run:
#──────────────────────────────────────────────────────────────────────────────
# Write Results and Save Plot

# Example 5a: Write results into a text file
test.welch(hp ~ gear, data = mtcars, write = "Welch-Test.txt")

# Example 5a: Write results into an Excel file
test.welch(hp ~ gear, data = mtcars, write = "Welch-Test.xlsx")
```

```
# Example 5b: Save plot as PNG fine
test.welch(hp ~ gear, data = mtcars, plot = TRUE,
            filename = "Welch-Test.png", width = 6, height = 5)

## End(Not run)
```

---

test.z *z-Test*

Description

This function performs the one-sample, two-sample, and paired-sample z-test and provides descriptive statistics, effect size measure, and a plot showing error bars for (difference-adjusted) confidence intervals with jittered data points.

Usage

```
test.z(...)

## Default S3 method:
test.z(data = NULL, ..., x = NULL, y = NULL, sigma = NULL, sigma2 = NULL,
       mu = 0, paired = FALSE, alternative = c("two.sided", "less", "greater"),
       hypo = FALSE, descript = TRUE, effsize = FALSE, conf.level = 0.95,
       digits = 2, p.digits = 3, as.na = NULL, plot = FALSE,
       geom = c("all", "bar", "point", "ci", "line", "jitter"), adjust = TRUE,
       filename = NULL, width = NA, height = NA, dpi = 600, write = NULL,
       append = TRUE, check = TRUE, output = TRUE)

## S3 method for class 'formula'
test.z(formula, data, sigma = NULL, sigma2 = NULL,
       alternative = c("two.sided", "less", "greater"), hypo = FALSE,
       descript = TRUE, effsize = FALSE, conf.level = 0.95, digits = 2,
       p.digits = 3, as.na = NULL, plot = FALSE,
       geom = c("all", "bar", "point", "ci", "jitter"), adjust = TRUE,
       filename = NULL, width = NA, height = NA, dpi = 600,
       write = NULL, append = TRUE, check = TRUE, output = TRUE, ...)
```

Arguments

`data`	a data frame. Note that data are specified either by using the argument `data` or by using the arguments `x` and `y`.
`...`	an expression indicating the variable names in `data`, e.g., `test.z(dat, x1, sigma = 2)`. Note that the operators `+`, `-`, `~`, `:`, `::`, and `!` can also be used to select variables, see 'Details' in the `df.subset` function.
`x`	a numeric vector of data values.
`y`	a numeric vector of data values.

formula	in case of two sample z-test (i.e., paired = FALSE), a formula of the form y ~ group where group is a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
sigma	a numeric vector indicating the population standard deviation(s). In case of two-sample z-test, equal standard deviations are assumed when specifying one value for the argument sigma; when specifying two values for the argument sigma, unequal standard deviations are assumed. Note that either argument sigma or argument sigma2 is specified.
sigma2	a numeric vector indicating the population variance(s). In case of two-sample z-test, equal variances are assumed when specifying one value for the argument sigma2; when specifying two values for the argument sigma, unequal variance are assumed. Note that either argument sigma or argument sigma2 is specified.
mu	a numeric value indicating the population mean under the null hypothesis. Note that the argument mu is only used when computing a one-sample z-test.
paired	logical: if TRUE, paired-sample z-test is computed.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
hypo	logical: if TRUE (default), null and alternative hypothesis are shown on the console.
descript	logical: if TRUE (default), descriptive statistics are shown on the console.
effsize	logical: if TRUE, effect size measure Cohen's d is shown on the console.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
digits	an integer value indicating the number of decimal places to be used for displaying descriptive statistics and confidence interval.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
plot	logical: if TRUE, a plot showing bar plots with error bars for confidence intervals is drawn. For additional plotting arguments, see Details in the help page of the function plot.misty.object.
geom	a character string or character vector indicating which plot elements to draw, i.e., "all" for drawing all plot elements, "bar" for drawing bars representing means for each groups, "point" for drawing points representing means for each group, "ci" for drawing error bars representing confidence intervals, "jitter" for drawing jittered data points, "line" for drawing a horizontal line at mu for the one-sample z-test or at 0 for the paired-sample z-test.
adjust	logical: if TRUE (default), difference-adjustment for the confidence intervals in a two-sample design is applied.
filename	a character string indicating the filename argument including the file extension in the ggsave function. Note that one of ".eps", ".ps", ".tex", ".pdf" (default), ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" needs to be specified as file extension in the file argument. Note that plots can only be saved when plot = TRUE.

width	a numeric value indicating the width argument (default is the size of the current graphics device) for the ggsave function.
height	a numeric value indicating the height argument (default is the size of the current graphics device) for the ggsave function.
dpi	a numeric value indicating the dpi argument (default is 600) for the ggsave function.
write	a character string naming a text file with file extension ".txt" (e.g., "Output.txt") for writing the output into a text file.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

Effect Size The Cohen's d reported when the argument effsize is set to TRUE is based on the population standard deviation specified in the argument sigma or the square root of the population variance specified in the argument sigma2.

- **One-Sample and Paired-Sample Design** In a one-sample and paired-sample design, Cohen's d is the mean of the difference scores divided by the population standard deviation of the (difference) scores equivalent to Cohen's d_z (Lakens, 2013).
- **Two-Sample Design** In a two-sample design, Cohen's d is the difference between means of the two groups of observations divided by either the population standard deviation when assuming and specifying equal standard deviations or the unweighted pooled population standard deviation when assuming and specifying unequal standard deviations.

Value

Returns an object of class misty.object, which is a list with following entries:

call	function call
type	type of analysis
sample	type of sample, i.e., one-, two-, or paired sample
formula	formula
data	data frame with the outcome and grouping variable
args	specification of function arguments
plot	ggplot2 object for plotting the results
result	result table

Author(s)

Takuya Yanagida

References

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs. *Frontiers in Psychology, 4*, 1-12. https://doi.org/10.3389/fpsyg.2013.00863

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

test.t, aov.b, aov.w, test.welch, cohens.d, ci.mean.diff, ci.mean

Examples

```
#————————————————————————————————————————————————————————————————
# One-Sample Design

# Example 1a: Two-sided one-sample z-test, population mean = 20, population SD = 6
test.z(mtcars, mpg, sigma = 6, mu = 20)

# Example 1b: Alternative specification without using the '...' argument
test.z(mtcars$mpg, sigma = 6, mu = 20)

# Example 1c: One-sided one-sample z-test, population mean = 20, population SD = 6,
# print Cohen's d
test.z(mtcars$mpg, sigma = 6, mu = 20, alternative = "greater", effsize = TRUE)

#————————————————————————————————————————————————————————————————
# Two-Sample Design

# Example 2a: Two-sided two-sample z-test, population SD = 6, equal SD assumption
test.z(mpg ~ vs, data = mtcars, sigma = 6)

# Example 2b: Alternative specification without using the '...' argument
test.t(x = df.subset(mtcars, mpg, subset = vs == 0),
       y = df.subset(mtcars, mpg, subset = vs == 1), sigma = 6)

# Example 2c: Two-sided two-sample z-test, population SD = 4 and 6, unequal SD assumption
test.z(mpg ~ vs, data = mtcars, sigma = c(4, 6))

# Example 2d: One-sided two-sample z-test, population SD = 4 and 6, unequal SD assumption
# print Cohen's d
test.z(mpg ~ vs, data = mtcars, sigma = c(4, 6), alternative = "greater", effsize = TRUE)

#————————————————————————————————————————————————————————————————
# Paired-Sample Design

# Example 3a: Two-sided paired-sample z-test, population SD of difference score = 1.2
test.z(mtcars, drat, wt, sigma = 1.2, paired = TRUE)

# Example 3b: Alternative specification without using the '...' argument
test.z(mtcars[, c("drat", "wt")], sigma = 1.2, paired = TRUE)
```

```
# Example 3c: Alternative specification without using the '...' argument
test.z(x = mtcars$drat, y = mtcars$wt, sigma = 1.2, paired = TRUE)

# Example 3d: One-sided paired-sample z-test, population SD of difference score = 1.2,
# print Cohen's d
test.z(x = mtcars$drat, y = mtcars$wt, sigma = 1.2, paired = TRUE, alternative = "greater",
       effsize = TRUE)

#———————————————————————————————————————————————————————————————————————
# Plot

# Example 4a: One-sample design
test.z(mtcars, mpg, sigma = 6, mu = 20, plot = TRUE)

# Example 4b: Two-sample design
test.z(mpg ~ vs, data = mtcars, sigma = 6, plot = TRUE)

# Example 4c: Paired-sample design
test.z(mtcars, drat, wt, sigma = 1.2, paired = TRUE, plot = TRUE)

# Example 4d: Plot results using the plot() function, use additional arguments
# see Details in the help page of the function plot.misty.object
object <- test.z(mpg ~ vs, data = mtcars, sigma = 6)
plot(object, geom = c("bar", "ci", "jitter"), jitter.alpha = 0.4, title = "Two-Sample z-Test")

#———————————————————————————————————————————————————————————————————————
# Create Plot Manually

# Load ggplot2 package
library(ggplot2)

# Example 4a: Two-sample z-test
ci.table <- ci.mean(mtcars, mpg, group = "vs", adjust = TRUE, output = FALSE)$result

ggplot(ci.table, aes(group, m)) +
  geom_bar(aes(group, m), stat = "summary", fun = "mean") +
  geom_errorbar(aes(group, m, ymin = low, ymax = upp), width = 0.1) +
  theme_bw()

# Example 4b: Paired-sample z-test
object <- test.z(mtcars, drat, wt, sigma = 1.2, paired = TRUE)

ggplot(data.frame(x = object$data$y - object$data$x), aes(x = 0L, y = x)) +
 geom_bar(data = object$result, aes(0, m.diff), stat = "summary", fun = "mean") +
 geom_errorbar(data = object$result, aes(0, m.diff, ymin = m.low, ymax = m.upp), width = 0.1) +
 geom_hline(yintercept = 0L, linetype = 3, linewidth = 0.8) +
 scale_x_continuous(name = "", limits = c(-2, 2)) +
 theme_bw()  +
 theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())

## Not run:
#———————————————————————————————————————————————————————————————————————
# Write Results and Save Plot
```

```
# Example 6a: Write results into a text file
test.z(mpg ~ vs, data = mtcars, sigma = 6, write = "z-Test.txt")

# Example 6b:  Write results into an Excel file
test.z(mpg ~ vs, data = mtcars, sigma = 6, write = "z-Test.xlsx")

# Example 4c: Two-Sample Design
test.z(mpg ~ vs, data = mtcars, sigma = 6, plot = TRUE, filename = "z-Test.png",
       width = 6, height = 5)

## End(Not run)
```

---

uniq *Extract Unique Elements and Count Number of Unique Elements*

Description

The function uniq returns a vector, list or data frame with duplicated elements removed. By default, the function prints a data frame with missing values omitted and unique elements sorted increasing. The function uniq.n counts the number of unique elements in a vector or for each column in a matrix or data frame. By default, missing values are omitted before counting the number of unique elements.

Usage

```
uniq(data, ..., na.rm = TRUE, sort = TRUE, decreasing = FALSE, digits = NULL,
     table = TRUE, write = NULL, append = TRUE, check = TRUE, output = TRUE)

uniq.n(data, ..., na.rm = TRUE, digits = NULL, check = TRUE)
```

Arguments

data	a vector, factor, matrix, or data frame.
...	an expression indicating the variable names in data, e.g., uniq(dat, x1, x2) for selecting the variables x1 and x2 from the data frame dat. Note that the operators +, -, ~, :, ::, and ! can also be used to select variables, see 'Details' in the df.subset function.
na.rm	logical: if TRUE (default), missing values are omitted before extracting unique elements. Note that missing values are always omitted when writing the output into an Excel file, i.e., na.rm = TRUE.
sort	logical: if TRUE (default), unique elements are sorted after extracting unique elements.
decreasing	logical: if TRUE, unique elements are sorted decreasing when specifying sort = TRUE

digits	an integer value indicating the number of decimal places to be used when rounding numeric values before extracting unique elements. By default, unique elements are extracted without rounding, i.e., digits = NULL.
table	logical: if TRUE (default), unique elements are printed in a data frame, if FALSE unique elements are printed in a list. Note that unique elements are always printed in a data frame when writing the output into an Excel file, i.e. table = TRUE.
write	a character string naming a file for writing the output into either a text file with file extension ".txt" (e.g., "Output.txt") or Excel file with file extension ".xlsx" (e.g., "Output.xlsx"). If the file name does not contain any file extension, an Excel file will be written.
append	logical: if TRUE (default), output will be appended to an existing text file with extension .txt specified in write, if FALSE existing text file will be overwritten.
check	logical: if TRUE (default), argument specification is checked.
output	logical: if TRUE (default), output is shown on the console.

Details

The function uniq is a wrapper function in the form of sort(unique(na.omit(x))), while the function uniq.n is a wrapper function in the form of length(unique(na.omit(x))).

Value

Returns an object of class misty.object when using the uniq function, which is a list with following entries:

call	function call
type	type of analysis
data	a vector, factor, matrix, or data frame
args	specification of function arguments
result	list with unique elements

or a vector with the count number of unique elements for a vector, factor or each column in a matrix or data frame when using the uniq.n function.

Author(s)

Takuya Yanagida

References

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

See Also

df.duplicated, df.unique

Examples

```
#═══════════════════════════════════════════════════════════════════════
# Extract Unique Elements, uniq() function

# Example 1a: Extract unique elements in a vector
uniq(airquality, Ozone)

# Example 1b: Extract unique elements in a vector, round elements
uniq(airquality, Wind, digits = 0)

# Example 1b: Extract unique elements in a vector, do not sort
uniq(airquality, Ozone, sort = FALSE)

# Example 1b: Extract unique elements in a vector, keep NA
uniq(airquality, Ozone, na.rm = FALSE)

# Example 2a: Extract unique elements in a data frame
uniq(airquality)

# Example 2a: Extract unique elements in list
uniq(airquality, table = FALSE)

#═══════════════════════════════════════════════════════════════════════
# Count Number of Unique Elements, uniq.n() function

# Example 3a: Count number of unique elements in a vector
uniq.n(airquality, Ozone)

# Example 3b: Count number of unique elements for each variable in a data frame
uniq.n(airquality)
```

`write.data` *Write Data File in Table Format, SPSS, Excel, or Stata DTA File*

Description

This function writes a (1) data file in CSV (`.csv`), DAT (`.dat`), or TXT (`.txt`) format using the `fwrite` function from the **data.table** package, (2) SPSS file (`.sav`) using the `write.sav` function, (3) Excel file (`.xlsx`) using the `write.xlsx` function, or a (4) Stata DTA file (`.dta`) using the `write.dta` function in the **misty** package. Note that the function `write.data` uses `","` for decimal point and a semicolon `";"` for the separator, while the function `write.data1` uses `"."` for decimal point and a comma `","` for the separator when writing a CSV file.

Usage

```
write.data(x, file = "Data.csv", sep = ";", dec = ",", na = "",
           row.names = FALSE, col.names = TRUE, check = TRUE, ...)

write.data1(x, file = "Data.csv", sep = ",", dec = ".", na = "",
            row.names = FALSE, col.names = TRUE, check = TRUE, ...)
```

Arguments

x	a matrix or data frame to be written.
file	a character string indicating the name of the data file with the file extension .csv, .dat, .txt, .sav, .xlsx, or .dta. Note that the function will select an appropriate write-function depending on the file extension.
sep	a character string indicating the field separator, i.e., string for the delimiter. By default, the write.data function uses a semicolon ";", while the function write.data1 function uses a comma "," for writing a CSV, DAT, or TXT data file
dec	a character string indicating the decimal separator, i.e., string for decimal points. By default, the write.data function uses a comma ",", while the function write.data1 function uses a decimal point "." for writing a CSV, DAT, or TXT data file.
na	a character string to use for missing values in the data. By default, a blank string "" is used.
row.names	logical: if FALSE, row names are written.
col.names	logical: if TRUE (default), column names are written.
check	logical: if TRUE (default), argument specification is checked.
...	additional arguments to pass to the fwrite write.sav, write.xlsx, or write.dta function, see Arguments section in the help pages.

Details

Comma-Separated Values (CSV) File The function write.data writes CSV files based on the Excel convention for CSV files in some Western European locales by default, i.e., ";" as delimiter and "," for decimal points. Depending on the language setting of the operating system of the computer, the arguments sep and dec need to be specified to "," and "." (see Example 1b). Alternatively, the function write.data1 that uses "," as delimiter and "." for decimal points by default can be used (see Example 1c).

Author(s)

Takuya Yanagida

References

Barrett, T., Dowle, M., Srinivasan, A., Gorecki, J., Chirico, M., Hocking, T., & Schwendinger, B. (2024). data.table: Extension of 'data.frame'. R package version 1.16.0. https://CRAN.R-project.org/package=data.table

Jeroen O. (2021). *writexl: Export Data Frames to Excel 'xlsx' Format*. R package version 1.4.0. https://CRAN.R-project.org/package=writexl

Wickham H, Miller E, Smith D (2023). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.5.3. https://CRAN.R-project.org/package=haven

See Also

read.data, read.sav, write.sav, write.xlsx, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:
# Example 1a: Write CSV data file, European format
write.data(mtcars, "European_CSV_Data.csv")

# Example 1b: Write CSV data file, American format
write.data(mtcars, "American_CSV_Data.csv", sep = ",", dec = ".")

# Example 1c: Write CSV data file, American format
write.data1(mtcars)

# Example 2: Write SPSS data file
write.data(mtcars, "SPSS_Data.sav")

# Example 3:  Write Excel data file
write.data(mtcars, "Excel_Data.xlsx")

# Example 4:  Write Stata data file
write.data(mtcars, "Stata_Data.dta")

## End(Not run)
```

`write.dta` *Write Stata DTA File*

Description

This function writes a data frame or matrix into a Stata data file.

Usage

```
write.dta(x, file = "Stata_Data.dta", version = 14, label = NULL,
          str.thres = 2045, adjust.tz = TRUE, check = TRUE)
```

Arguments

`x`	a matrix or data frame to be written in Stata, vectors are coerced to a data frame.
`file`	a character string naming a file with or without file extension '.dta', e.g., `"Stata_Data.dta"` or `"Stata_Data"`.
`version`	Stats file version to use. Supports versions 8-15.
`label`	dataset label to use, or `NULL`. Defaults to the value stored in the "label" attribute pf data. Must be <= 80 characters.
`str.thres`	any character vector with a maximum length greater than `str.thre` bytes will be stored as a long string `strL` instead of a standard string `str` variable if `version` is greater or equal 13.
`adjust.tz`	this argument controls how the timezone of date-time values is treated when writing, see 'Details' in the in the `write_dta` function in the `havan` package.
`check`	logical: if `TRUE` (default), variable attributes specified in the argument `var.attr` is checked.

Note

This function is a modified copy of the `read_dta()` function in the **haven** package by Hadley Wickham, Evan Miller and Danny Smith (2023).

Author(s)

Takuya Yanagida

References

Wickham H, Miller E, Smith D (2023). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.5.3. https://CRAN.R-project.org/package=haven

See Also

`read.data`, `write.data`, `read.sav`, `write.sav`, `write.xlsx`, `read.dta`, `read.mplus`, `write.mplus`

Examples

```
## Not run:

# Example 1: Write data frame 'mtcars' into the State data file 'mtcars.dta'
write.dta(mtcars, "mtcars.dta")

## End(Not run)
```

`write.mplus` *Write Mplus Data File and Mplus Input Template*

Description

This function writes a matrix or data frame to a tab-delimited file without variable names, a Mplus input template, and a text file with variable names. Note that only numeric variables are allowed, i.e., non-numeric variables will be removed from the data set. Missing data will be coded as a single numeric value.

Usage

```
write.mplus(x, file = "Mplus_Data.dat", data = TRUE, input = TRUE,
            var = FALSE, na = -99, check = TRUE)
```

Arguments

`x`	a matrix or data frame to be written to a tab-delimited file.
`file`	a character string naming a file with or without the file extension '.dat', e.g., `"Mplus_Data.dat"` or `"Mplus_Data"`.
`data`	logical: if `TRUE` (default), Mplus data file is written in a text file named according to the argument`file`.

input	logical: if TRUE (default), Mplus input template is written in a text file named according to the argumentfile with the extension _INPUT.inp.
var	logical: if TRUE, variable names are written in a text file named according to the argumentfile with the extension _VARNAMES.txt.
na	a numeric value or character string representing missing values (NA) in the data set.
check	logical: if TRUE (default), argument specification is checked.

Value

Returns a character string indicating the variable names for the Mplus input file.

Author(s)

Takuya Yanagida

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.data, write.data, read.sav, write.sav, write.xlsx, read.dta, write.dta, read.mplus

Examples

```
## Not run:

# Example 1: Write Mplus Data File and a Mplus input template
write.mplus(mtcars)

# Example 2: Write Mplus Data File "mtcars.dat" and a Mplus input template "mtcars_INPUT.inp",
# missing values coded with -999,
# write variable names in a text file called "mtcars_VARNAMES.inp"
write.mplus(mtcars, file = "mtcars.dat", var = TRUE, na = -999)

## End(Not run)
```

---

write.result *Write Results of a misty Object into an Excel file*

Description

This function writes the results of a misty.object) into an Excel file.

Usage

```
write.result(x, file = "Results.xlsx", digits = NULL, p.digits = NULL,
             icc.digits = NULL, r.digits = NULL, ess.digits = NULL,
             mcse.digits = NULL, check = TRUE)
```

Arguments

`x`	misty object (`misty.object`) resulting from a misty function supported by the `write.result` function (see 'Details').
`file`	a character string naming a file with or without file extension '.xlsx', e.g., `"Results.xlsx"` or `"Results"`.
`digits`	an integer value indicating the number of decimal places digits to be used for displaying results.
`p.digits`	an integer indicating the number of decimal places to be used for displaying *p*-values.
`icc.digits`	an integer indicating the number of decimal places to be used for displaying intraclass correlation coefficients.
`r.digits`	an integer value indicating the number of decimal places to be used for displaying R-hat values, item-total correlation standardized factor loadings, coefficient alpha, and coefficient omega.
`ess.digits`	an integer value indicating the number of decimal places to be used for displaying effective sample sizes.
`mcse.digits`	an integer value indicating the number of decimal places to be used for displaying Monte Carlo standard errors.
`check`	logical: if `TRUE` (default), argument specification is checked.

Author(s)

Takuya Yanagida

`write.sav` *Write SPSS File*

Description

This function writes a data frame or matrix into a SPSS file by either using the `write_sav()` function in the **haven** package by Hadley Wickham and Evan Miller (2019) or the free software *PSPP*.

Usage

```
write.sav(x, file = "SPSS_Data.sav", var.attr = NULL, pspp.path = NULL,
          digits = 2, write.csv = FALSE, sep = c(";", ","), na = "",
          write.sps = FALSE, check = TRUE)
```

Arguments

`x`	a matrix or data frame to be written in SPSS, vectors are coerced to a data frame.
`file`	a character string naming a file with or without file extension '.sav', e.g., `"SPSS_Data.sav"` or `"SPSS_Data"`.
`var.attr`	a matrix or data frame with variable attributes used in the SPSS file, only 'variable labels' (column name `label`), 'value labels' column name `values`, and 'user-missing values' column name `missing` are supported (see 'Details').
`pspp.path`	a character string indicating the path where the PSPP folder is located on the computer, e.g.`C:/Program Files/PSPP/`.
`digits`	an integer value indicating the number of decimal places shown in the SPSS file for non-integer variables.
`write.csv`	logical: if `TRUE`, CSV file is written along with the SPSS file.
`sep`	a character string for specifying the CSV file, either `";"` for the separator and `"."` for the decimal point (default, i.e. equivalent to `write.csv2`) or `"."` for the decimal point and `","` for the separator (i.e. equivalent to `write.csv`), must be one of both `";"` (default) or `","`.
`na`	a character string for specifying missing values in the CSV file.
`write.sps`	logical: if `TRUE`, SPSS syntax is written along with the SPSS file when using PSPP.
`check`	logical: if `TRUE`, variable attributes specified in the argument `var.attr` is checked.

Details

If arguments `pspp.path` is not specified (i.e., `pspp.path = NULL`), `write_sav()` function in the **haven** is used. Otherwise the object `x` is written as CSV file, which is subsequently imported into SPSS using the free software *PSPP* by executing a SPSS syntax written in R. Note that *PSPP* needs to be installed on your computer when using the `pspp.path` argument.

A SPSS file with 'variable labels', 'value labels', and 'user-missing values' is written by specifying the `var.attr` argument. Note that the number of rows in the matrix or data frame specified in `var.attr` needs to match with the number of columns in the data frame or matrix specified in `x`, i.e., each row in `var.attr` represents the variable attributes of the corresponding variable in `x`. In addition, column names of the matrix or data frame specified in `var.attr` needs to be labeled as `label` for 'variable labels, `values` for 'value labels', and `missing` for 'user-missing values'.

Labels for the values are defined in the column `values` of the matrix or data frame in `var.attr` using the equal-sign (e.g., `0 = female`) and are separated by a semicolon (e.g., `0 = female; 1 = male`).

User-missing values are defined in the column `missing` of the matrix or data frame in `var.attr`, either specifying one user-missing value (e.g., `-99`) or more than one but up to three user-missing values separated by a semicolon (e.g., `-77; -99`.

Note

Part of the function using *PSPP* was adapted from the `write.pspp()` function in the **miceadds** package by Alexander Robitzsch, Simon Grund and Thorsten Henke (2019).

Author(s)

Takuya Yanagida

References

GNU Project (2018). *GNU PSPP for GNU/Linux* (Version 1.2.0). Boston, MA: Free Software Foundation. https://www.gnu.org/software/pspp/

Wickham H., & Miller, E. (2019). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.2.0.

Robitzsch, A., Grund, S., & Henke, T. (2019). *miceadds: Some additional multiple imputation functions, especially for mice*. R package version 3.4-17.

See Also

read.data, write.data, read.sav, write.xlsx, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:

dat <- data.frame(id = 1:5,
                  gender = c(NA, 0, 1, 1, 0),
                  age = c(16, 19, 17, NA, 16),
                  status = c(1, 2, 3, 1, 4),
                  score = c(511, 506, 497, 502, 491))

# Example 1: Write SPSS file using the haven package
write.sav(dat, file = "Dataframe_haven.sav")

# Example 2: Write SPSS file using PSPP,
# write CSV file and SPSS syntax along with the SPSS file
write.sav(dat, file = "Dataframe_PSPP.sav", pspp.path = "C:/Program Files/PSPP",
          write.csv = TRUE, write.sps = TRUE)

# Example 3: Specify variable attributes
# Note that it is recommended to manually specify the variables attritbues in a CSV or
# Excel file which is subsequently read into R
attr <- data.frame(# Variable names
                   var = c("id", "gender", "age", "status", "score"),
                   # Variable labels
                   label = c("Identification number", "Gender", "Age in years",
                             "Migration background", "Achievement test score"),
                   # Value labels
                   values = c("", "0 = female; 1 = male", "",
                             "1 = Austria; 2 = former Yugoslavia; 3 = Turkey; 4 = other",
                              ""),
                   # User-missing values
                   missing = c("", "-99", "-99", "-99", "-99"))

# Example 4: Write SPSS file with variable attributes using the haven package
write.sav(dat, file = "Dataframe_haven_Attr.sav", var.attr = attr)
```

```
# Example 5: Write SPSS with variable attributes using PSPP
write.sav(dat, file = "Dataframe_PSPP_Attr.sav", var.attr = attr,
          pspp.path = "C:/Program Files/PSPP")

## End(Not run)
```

`write.xlsx` *Write Excel File*

Description

This function calls the `write_xlsx()` function in the **writexl** package by Jeroen Ooms to write an Excel file (.xlsx).

Usage

```
write.xlsx(x, file = "Excel_Data.xlsx", col.names = TRUE, format = FALSE,
           use.zip64 = FALSE, check = TRUE)
```

Arguments

`x`	a matrix, data frame or (named) list of matrices or data frames that will be written in the Excel file.
`file`	a character string naming a file with or without file extension '.xlsx', e.g., `"My_Excle.xlsx"` or `"My_Excel"`.
`col.names`	logical: if `TRUE`, column names are written at the top of the Excel sheet.
`format`	logical: if `TRUE`, column names in the Excel file are centered and bold.
`use.zip64`	logical: if `TRUE`, zip64 to enable support for 4GB+ Excel files is used.
`check`	logical: if `TRUE` (default), argument specification is checked.

Details

This function supports strings, numbers, booleans, and dates.

Note

The function was adapted from the `write_xlsx()` function in the **writexl** package by Jeroen Ooms (2021).

Author(s)

Takuya Yanagida

References

Jeroen O. (2021). *writexl: Export Data Frames to Excel 'xlsx' Format*. R package version 1.4.0. https://CRAN.R-project.org/package=writexl

See Also

read.data, write.data, read.sav, write.sav, read.dta, write.dta, read.mplus, write.mplus

Examples

```
## Not run:
# Example 1: Write Excel file (.xlsx)
write.xlsx(mtcars, file = "mtcars.xlsx")

# Example 2: Write Excel file with multiple sheets (.xlsx)
write.xlsx(list(cars = cars, mtcars = mtcars), file = "Excel_Sheets.xlsx")
## End(Not run)
```

Index

∗ **datasets**
data.items, 135

aov.b, 4, *7*, *12*, *428*, *439*, *444*, *449*
aov.w, *7*, 8, *92*, *439*, *449*
as.na, *353*, *355*, *357*, *359*, *361*, *363*, *370*
as.na *(*na.as*)*, 348

blimp, 13, *22*, *26*, *31*, *33*
blimp.bayes, *16*, 17, *26*, *31*, *33*, *37*
blimp.plot, *16*, *22*, 22, *26*, *31*, *33*, *37*
blimp.print, *15, 16*, *22*, *26*, 29, *33*, *35*, *37*
blimp.run, *16*, *22*, *26*, *31*, 32, *37*
blimp.update, *16*, *22*, *26*, *31*, *33*, 34
boot.bs, 38, *375*

center, 42, *237*
cfa.satcor *(*na.satcor*)*, 363
check.collin, 50, *55*, *60*, *414, 415*
check.outlier, *51*, 53, *60*
check.resid, *55*, 57, *424*, *431*, *435*
chr.color, 62, *64*, *66–69*, *129*, *140*, *144*, *187*, *202*, *209*, *296*, *305*, *326*
chr.grep, *63*, 63, *66–69*
chr.grepl, *63, 64*, *66–69*
chr.grepl *(*chr.grep*)*, 63
chr.gsub, *63, 64*, 65, *67–69*
chr.omit, *63, 64*, *66*, 66, *68, 69*
chr.trim, *63, 64*, *66, 67*, 67, *69*
chr.trunc, *63, 64*, *66–68*, 69
ci.cor, 70, *81*, *84*, *93*, *95*, *101*, *103*, *232*
ci.mean, *7*, *12*, *78*, 80, *88*, *99*, *103*, *138*, *439*, *444*, *449*
ci.mean.diff, *7*, *12*, *78*, *84*, 86, *92*, *99*, *103*, *138*, *439*, *444*, *449*
ci.mean.w, 89
ci.median, *88*, *92*, *95*, *99*, *103*, *138*
ci.median *(*ci.mean*)*, 80
ci.prop, *78*, *84*, *88*, *92*, 92, *95*, *99*, *103*, *138*
ci.prop.diff, *95*, 97, *103*, *138*
ci.sd, *78*, *84*, *88*, *92*, *95*, *99*, *138*
ci.sd *(*ci.var*)*, 100
ci.var, *78*, *84*, *88*, *92*, *95*, *99*, 100, *138*
clear, 105
cluster.rwg, *47*, 106
cluster.scores, *47*, *107*, 108, *230*, *307*
coding, *47*, 110, *237*, *396*
coeff.robust, 112, *400*, *414, 415*, *424*, *431*
coeff.std, *117*, 118, *166*, *168*, *414, 415*
cohens.d, *7*, *12*, 123, *131*, *173*, *438, 439*, *442*, *444*, *449*
cor.matrix, *78*, *126*, 128, *173*, *409*, *413*, *415*
cor.test, *409*
crosstab, 132, *138*, *175*

data.items, 135
descript, *84*, *88*, *92*, *95*, *99*, *103*, *134*, 136, *175*, *312*, *412*, *415*
df.check, 140, *142*, *144*, *147*, *149*, *151*, *153–155*, *157*
df.duplicated, *141*, 141, *144*, *147*, *149*, *151*, *153–155*, *157*, *452*
df.head, *141, 142*, 143, *147*, *149*, *151*, *153–155*, *157*
df.long, *141, 142*, *144*, 145, *149*, *151*, *153–155*, *157*
df.merge, *141, 142*, *144*, *147*, 148, *151*, *153–155*, *157*
df.move, *141, 142*, *144*, *147*, *149*, 150, *153–155*, *157*
df.rbind, *141, 142*, *144*, *147*, *149*, *151*, 152, *154, 155*, *157*
df.rename, *141, 142*, *144*, *147*, *149*, *151*, *153*, 153, *155*, *157*
df.sort, *141, 142*, *144*, *147*, *149*, *151*, *153, 154*, 154, *157*
df.subset, *42*, *71*, *81*, *87*, *90*, *92*, *98*, *100*, *106*, *108*, *123*, *128*, *133*, *137*, *141, 142*, *144, 145*, *147*, *149–151*, *153–155*, 156, *171*, *174*, *180*, *185*,

201, 205, 216, 224, 227, 229, 232, 235, 289, 293, 303, 309, 316, 324, 330, 348, 351, 354, 356, 358, 360, 362, 365, 395, 411, 419, 433, 438, 446, 451
df.tail, 141, 142, 147, 149, 151, 153–155, 157
df.tail (df.head), 143
df.unique, 141, 144, 147, 149, 151, 153–155, 157, 452
df.unique (df.duplicated), 141
df.wide, 141, 142, 144, 149, 151, 153–155, 157
df.wide (df.long), 145
difftest.chibarsq, 159
dominance, 164, 168
dominance.manual, 166, 167

effsize, 126, 131, 171

format.POSIXct, 401
freq, 134, 138, 173

growth.satcor (na.satcor), 363

indirect, 176, 321, 323
item.alpha, 180, 193, 203, 228, 230, 234
item.cfa, 181, 183, 184, 198, 203, 211, 222, 225, 226, 230, 234, 300, 351
item.dfi, 195, 379
item.distract, 201, 234
item.invar, 183, 204, 222, 226
item.noninvar, 211, 215
item.omega, 183, 193, 203, 224, 226, 228, 230, 234, 291, 334
item.reverse, 47, 112, 183, 203, 226, 227, 234, 237, 396
item.scores, 47, 109, 183, 193, 203, 226, 228, 229, 234
item.stats, 203, 231

kurtosis, 435
kurtosis (skewness), 410

lagged, 235
lavaan.satcor (na.satcor), 363
libraries, 238
library, 238
lm, 51

modcomp, 239
mplus, 246, 253, 258, 270, 281, 283, 287
mplus.bayes, 248, 249, 258, 270, 281, 283, 287
mplus.lca, 248, 253, 254, 265, 270, 281, 283, 287
mplus.lca.summa, 258, 259
mplus.plot, 248, 253, 258, 266, 281, 283, 287
mplus.print, 247, 248, 253, 258, 270, 276, 283, 285, 287
mplus.run, 248, 253, 258, 265, 270, 281, 282, 287
mplus.update, 248, 253, 258, 270, 281, 283, 284
multilevel.alpha, 288, 334
multilevel.cfa, 291, 292, 315, 319, 328, 334
multilevel.cor, 131, 291, 300, 303, 315, 319, 328, 334, 342, 346, 413
multilevel.descript, 109, 134, 138, 175, 291, 300, 307, 309, 315, 319, 328, 334, 342, 346
multilevel.fit, 291, 300, 314, 328, 334
multilevel.icc, 109, 131, 307, 310–312, 316, 342, 346
multilevel.indirect, 179, 320, 342, 346
multilevel.invar, 211, 222, 291, 300, 315, 323, 334
multilevel.omega, 291, 300, 315, 328, 330
multilevel.r2, 336, 346, 413
multilevel.r2.manual, 344

na.as, 348, 353, 355, 357, 359, 361, 363, 370
na.auxiliary, 126, 131, 349, 350, 355, 357, 359, 361, 363, 370
na.coverage, 349, 353, 354, 357, 359, 361, 363, 370
na.descript, 134, 138, 175, 349, 353, 355, 356, 359, 361, 363, 370
na.indicator, 349, 353, 355, 357, 358, 361, 363, 370
na.pattern, 349, 353, 355, 357, 359, 359, 363, 370
na.prop, 349, 353, 355, 357, 359, 361, 362, 370
na.satcor, 363
na.test, 349, 353, 355, 357, 359, 361, 363, 365

p.adjust, 9, 129, 305, 419, 430, 433, 438, 443

plot.misty.object, *6*, *10*, *58*, *337*, *346*, 370, *423*, *427*, *431*
print.misty.object, 385
prop.test, *409*

rbind, *152*
read.data, 386, *389*, *391, 392*, *394*, *454*, *456, 457*, *460*, *462*
read.dta, *387*, 388, *391, 392*, *394*, *454*, *456, 457*, *460*, *462*
read.mplus, *248*, *253*, *258*, *265*, *270*, *281*, *283*, *287*, *387*, *389*, 390, *392*, *394*, *454*, *456, 457*, *460*, *462*
read.sav, *387*, *389*, *391*, 391, *394*, *454*, *456, 457*, *460*, *462*
read.xlsx, 392
rec, *47*, *112*, *228*, *237*, 395
require, *238*
restart, *105*, 397
robust.lmer, 398

script.close, *402, 403*, *405*
script.close *(*script.open*)*, 403
script.copy, 401, *403, 404*
script.new, *402*, 402, *405*
script.open, *402, 403*, 403, *405*
script.save, *402–405*
script.save *(*script.open*)*, 403
sem.satcor *(*na.satcor*)*, 363
setsource, *105*, *402–404*, 405
sim.lavaan, 406
size.cor *(*size.mean*)*, 408
size.mean, 408
size.prop *(*size.mean*)*, 408
skewness, 410, *435*
summa, *400*, 413

test.bm, 418
test.homo, 422, *431*, *435*
test.levene, *7*, 426, *435*, *439*, *444*
test.linear, *424*, 429, *435*
test.norm, *424*, *431*, 433
test.t, *7*, *12*, *84*, *88*, *92*, *126*, *409*, *421*, *428*, 437, *444*, *449*
test.welch, *421*, *428*, *439*, 442, *449*
test.z, *7*, *12*, *84*, *88*, *92*, *126*, *421*, *439*, *444*, 446

uniq, 451

write.data, *387*, *389*, *391, 392*, *394*, 453, *456, 457*, *460*, *462*
write.data1 *(*write.data*)*, 453
write.dta, *387*, *389*, *391, 392*, *394*, *454*, 455, *457*, *460*, *462*
write.mplus, *248*, *253*, *258*, *265*, *270*, *281*, *283*, *287*, *387*, *389*, *391, 392*, *394*, *454*, *456*, 456, *460*, *462*
write.result, *117*, *131*, *134*, *166*, *168*, *175*, *183*, *226*, *291*, *307*, *312*, *355*, *357*, *361*, 457
write.sav, *387*, *389*, *391, 392*, *394*, *454*, *456, 457*, 458, *462*
write.xlsx, *387*, *389*, *391, 392*, *394*, *454*, *456, 457*, *460*, 461