Package 'colleyRstats'

September 2, 2026

Type Package

Title Functions to Streamline Statistical Analysis and Reporting

Version 0.2.0

Description Built upon popular R packages such as 'ggstatsplot' and 'ARTool', this collection offers a wide array of tools for simplifying reproducible analyses, generating high-quality visualizations, and producing 'APA'-compliant outputs. The primary goal of this package is to significantly reduce repetitive coding efforts, allowing you to focus on interpreting results. Whether you're dealing with ANOVA assumptions, reporting effect sizes, or creating publication-ready visualizations, this package makes these tasks easier.

URL https://github.com/M-Colley/colleyRstats, https://m-colley.github.io/colleyRstats/

BugReports https://github.com/M-Colley/colleyRstats/issues

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.2.0), ggplot2 (>= 3.4.0)

Imports dplyr (>= 1.1.0), effectsize, ggpmisc (>= 0.5.0), ggsignif, ggstatsplot (>= 1.0.0), grid, lifecycle, purrr, rlang, see, stats, statsExpressions (>= 2.0.0), stringr, tidyr, utils

Suggests afex, ARTool (>= 0.11.0), av, BayesFactor, bayestestR, Cairo, car, clipr, conflicted, DT, dunn.test, easystats, emmeans, emoa, flexdashboard, FSA, ggtext, glmmTMB, Hmisc, knitr, lme4, lmerTest, moocore, nnet, nparLD, ordinal, parameters, patchwork, pkgload, psych, readxl, renv, report, reporttools, rmarkdown, rstantools, rstatix, scales, sjPlot, stargazer, styler, testthat (>= 3.3.0), tibble, withr, writexl, xtable

Config/testthat/edition 3

Config/Needs/check anthonynorth/roxyglobals

Config/roxyglobals/unique TRUE

Config/testthat/parallel false

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Mark Colley [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5207-5029>)

Maintainer Mark Colley <mark.colley@yahoo.de>

Repository CRAN

Date/Publication 2026-09-02 12:50:02 UTC

Contents

add_pareto_emoa_column . . . 3
add_pareto_moocore_column . . . 4
analyze_and_report . . . 6
animate_mobo2 . . . 7
assumption_methods_text . . . 9
checkAssumptionsForAnova . . . 9
check_homogeneity_by_group . . . 10
check_normality_by_group . . . 11
check_questionnaire . . . 12
cite_methods . . . 13
classify_outcome . . . 14
colleyRstats_setup . . . 15
colley_theme . . . 17
debug_contr_error . . . 18
define_questionnaire . . . 19
define_result_macro . . . 21
emit_name_macros . . . 22
emit_overleaf . . . 22
expand_latex_macros . . . 23
figure_base_size . . . 24
fit_recommended . . . 25
generateEffectPlot . . . 27
generateMoboPlot . . . 29
generateMoboPlot2 . . . 31
ggbetweenstatsWithPriorNormalityCheck . . . 33
ggbetweenstatsWithPriorNormalityCheckAsterisk . . . 35
ggwithinstatsWithPriorNormalityCheck . . . 36
ggwithinstatsWithPriorNormalityCheckAsterisk . . . 38
latexify_report . . . 39
latex_escape . . . 41
latex_preamble . . . 41
list_questionnaires . . . 42
na.zero . . . 42
normalize . . . 43
not_empty . . . 44

not_in . . . 44
n_fun . . . 45
pathPrep . . . 45
questionnaire_items . . . 46
recommend_test . . . 47
remove_outliers_REI . . . 48
replace_values . . . 49
reportART . . . 50
reportArtCon . . . 52
reportArtConTable . . . 53
reportCLMM . . . 56
reportDunnTest . . . 57
reportDunnTestTable . . . 58
reportggstatsplot . . . 60
reportggstatsplotPostHoc . . . 62
reportGLMM . . . 63
reportMeanAndSD . . . 65
reportNparLD . . . 66
reportNPAV . . . 67
report_all . . . 68
reshape_data . . . 69
reverse_code . . . 71
rFromNPAV . . . 72
rFromWilcox . . . 72
rFromWilcoxAdjusted . . . 73
save_paper_figure . . . 74
score_questionnaire . . . 75
score_reliability . . . 77
stat_sum_df . . . 79
summarize_sickness . . . 80
use_colleyrstats_sty . . . 81
use_study_project . . . 81

Index **84**

`add_pareto_emoa_column`

Add `PARETO_EMOA` *Column to a Data Frame*

Description

This function calculates the Pareto front using emoa for a given set of objectives in a data frame and adds a new column, `PARETO_EMOA`, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_emoa_column(data, objectives, maximise = FALSE)
```

Arguments

`data`	A data frame containing the data, including the objective columns.
`objectives`	A character vector specifying the names of the objective columns in `data`. These columns should be numeric and will be used to calculate the Pareto front.
`maximise`	Direction of optimisation. `FALSE` (the default) treats every objective as one to be *minimised*, which is what **emoa** does natively. Pass `TRUE` when larger is better for every objective – as it is for trust, acceptance, perceived safety and most other rating-scale outcomes – or a logical vector with one entry per objective for a mixed problem, e.g. `c(TRUE, TRUE, FALSE)` to maximise the first two and minimise the third. Objectives flagged `TRUE` are negated internally, so you no longer need to pass negated copies of your own columns.

Value

A data frame with the same columns as `data`, along with an additional column, `PARETO_EMOA`, which is `TRUE` for rows that are on the Pareto front and `FALSE` otherwise.

See Also

`add_pareto_moocore_column()`, which answers the same question via **moocore** and accepts the same `maximise` argument.

Examples

```
# Define objective columns
objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

# Example data frame
main_df <- data.frame(
  trust = runif(10),
  predictability = runif(10),
  perceivedSafety = runif(10),
  Comfort = runif(10)
)

# Add the Pareto front column (minimising, the default)
main_df <- add_pareto_emoa_column(data = main_df, objectives)
head(main_df)

# All four objectives are ratings where higher is better
main_df <- add_pareto_emoa_column(main_df, objectives, maximise = TRUE)
```

`add_pareto_moocore_column` *Add* `PARETO_MOOCORE` *Column to a Data Frame*

Description

This function calculates the Pareto front using moocore for a given set of objectives in a data frame and adds a new column, PARETO_MOOCORE, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_moocore_column(data, objectives, maximise = FALSE)
```

Arguments

data	A data frame containing the data, including the objective columns.
objectives	A character vector specifying the names of the objective columns in data. These columns should be numeric and will be used to calculate the Pareto front.
maximise	Direction of optimisation, passed through to moocore::is_nondominated(). FALSE (the default) treats every objective as one to be *minimised*. Pass TRUE when larger is better for every objective – as it is for trust, acceptance, perceived safety and most other rating-scale outcomes – or a logical vector with one entry per objective for a mixed problem, e.g. c(TRUE, TRUE, FALSE) to maximise the first two and minimise the third. This removes the need to pass negated copies of your own columns.

Value

A data frame with the same columns as data, along with an additional column, PARETO_MOOCORE, which is TRUE for rows that are on the Pareto front and FALSE otherwise.

See Also

add_pareto_emoa_column(), which answers the same question via **emoa** and accepts the same maximise argument.

Examples

```
# Define objective columns
objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

# Example data frame
main_df <- data.frame(
  trust = runif(10),
  predictability = runif(10),
  perceivedSafety = runif(10),
  Comfort = runif(10)
)

# Add the Pareto front column (minimising, the default)
main_df <- add_pareto_moocore_column(data = main_df, objectives)
head(main_df)

# All four objectives are ratings where higher is better
```

```
main_df <- add_pareto_moocore_column(main_df, objectives, maximise = TRUE)

# Mixed: maximise the ratings, minimise a workload score
main_df$workload <- runif(10)
main_df <- add_pareto_moocore_column(
  main_df,
  c(objectives, "workload"),
  maximise = c(TRUE, TRUE, TRUE, TRUE, FALSE)
)
```

`analyze_and_report` *Analyze one dependent variable and produce everything a paper needs*

Description

One-call pipeline for a single dependent variable: checks the assumptions (producing a ready-made methods sentence via `assumption_methods_text()`), builds the matching **ggstatsplot** figure with automatic parametric/non-parametric selection, reports the omnibus test via `reportggstatsplot()`, and – for more than two groups – reports the significant post-hoc comparisons via `reportggstatsplotPostHoc()`.

Usage

```
analyze_and_report(
  data,
  dv,
  iv,
  design = c("between", "within"),
  ylab = dv,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_to = NULL
)
```

Arguments

`data`	the data frame
`dv`	the dependent variable (column name as string)
`iv`	the independent variable (column name as string); coerced to a factor if it is not one already
`design`	`"between"` for between-subjects data (default) or `"within"` for repeated measures
`ylab`	label for the dependent variable; defaults to `dv`
`xlabels`	optional labels for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)
`sink_to`	optional path of a `.tex` file; the methods sentence, omnibus result, and post-hoc sentences are written there so a manuscript can `\input{}` them

Value

Invisibly returns a list with components `plot` (the ggplot), `methods` (assumption-check sentence), `text` (omnibus result), `posthoc` (post-hoc sentences, or `NULL` for two groups), and `sentences` (all text combined, in manuscript order).

Examples

```
result <- analyze_and_report(mtcars, dv = "mpg", iv = "cyl")
result$plot
```

`animate_mobo2` *Animate a Multi-objective Optimization Plot*

Description

Renders `generateMoboPlot2()` one iteration at a time and encodes the frames into a video, so a talk or a supplement can show an optimisation run building up rather than only its end state. Frame *i* is the same plot restricted to the data up to iteration *i*: the mean points, their bootstrapped intervals, the fitted line and its equation all move as the run proceeds.

Usage

```
animate_mobo2(
  data,
  x = "Iteration",
  y,
  filename,
  ...,
  fps = 5,
  end_pause = 2,
  width = 8,
  height = 5,
  dpi = 150,
  label_iterations = TRUE
)
```

Arguments

`data`	A data frame holding the run, as for `generateMoboPlot2()`.
`x`	A string naming the iteration column in `data`. Default `"Iteration"`.
`y`	A string naming the objective column in `data`.
`filename`	Path of the video to write. Its extension chooses the container: `.mp4`, `.gif`, `.mov`, `.webm` – whatever the `av` package's FFmpeg build can encode.
`...`	Further arguments for `generateMoboPlot2()`, e.g. `phaseCol`, `fillColourGroup`, `ytext`, `legendPos`, `fillLabels`.

`fps`	Frames per second. One frame is drawn per iteration, so this is equally the number of iterations shown per second. Default 5.
`end_pause`	Seconds to hold the final frame, so the finished plot can be read before the video ends or loops. Default 2; use 0 for no hold.
`width, height`	Frame size in inches. Default 8 x 5.
`dpi`	Resolution. Default 150, so the default frame is 1200 x 750 px.
`label_iterations`	Whether to caption each frame with the iteration it shows. Default `TRUE`, which writes the plot's subtitle.

Details

The plot is built once from the complete data, and a frame only ever hides rows. That is what keeps the axes, the sampling/optimisation guides and the legend still while the data grows – and it is also the only way the early frames can be drawn at all, since `generateMoboPlot2()` requires both phases to be present and the first iterations are all sampling.

Value

Invisibly returns `filename`.

See Also

`generateMoboPlot2()` for the static plot.

Examples

```
# Kept deliberately tiny: every frame is a full ggplot render, so the cost
# of the example is set by the number of iterations, not by the frame size.
if (requireNamespace("av", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    Iteration   = rep(1:3, each = 4),
    ConditionID = rep(rep(c("A", "B"), each = 2), 3),
    Phase       = rep(c("Sampling", "Optimization"), times = c(2 * 4, 4))
  )
  df$score <- 0.3 + 0.05 * df$Iteration + stats::rnorm(nrow(df), sd = 0.05)

  animate_mobo2(
    df,
    y = "score", ytext = "Score",
    filename = file.path(tempdir(), "mobo.mp4"),
    width = 4, height = 2.5, dpi = 96, end_pause = 0
  )
}
```

`assumption_methods_text` *Methods-section sentence justifying the test selection*

Description

Runs the group-wise Shapiro-Wilk normality check (and optionally Levene's test for homogeneity of variances) and turns the outcome into a ready-made methods-section sentence, including the relevant statistics. This is the justification reviewers expect next to the choice of a parametric or non-parametric test.

Usage

```
assumption_methods_text(data, x, y, include_homogeneity = FALSE)
```

Arguments

`data`	the data frame
`x`	the grouping variable (column name as string)
`y`	the dependent variable (column name as string)
`include_homogeneity`	whether to also report Levene's test. Useful for between-subjects designs. Default `FALSE`.

Value

Invisibly returns the sentence(s) as a single string; the text is also emitted via `message()`.

Examples

```
set.seed(1)
d <- data.frame(g = rep(c("A", "B"), each = 20), v = rnorm(40))
assumption_methods_text(d, x = "g", y = "v")
```

`checkAssumptionsForAnova` *Check the assumptions for an ANOVA with a variable number of factors: Normality and Homogeneity of variance assumption.*

Description

Check the assumptions for an ANOVA with a variable number of factors: Normality and Homogeneity of variance assumption.

Usage

```
checkAssumptionsForAnova(data, y, factors)

check_assumptions_anova(data, y, factors)
```

Arguments

data	the data frame
y	The dependent variable for which assumptions should be checked
factors	A character vector of factor names

Value

A message indicating whether to use parametric or non-parametric ANOVA

Naming

`check_assumptions_anova()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`checkAssumptionsForAnova()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

main_df <- data.frame(
  tlx_mental      = rnorm(40),
  Video           = factor(rep(c("A", "B"), each = 20)),
  DriverPosition  = factor(rep(c("Left", "Right"), times = 20))
)

checkAssumptionsForAnova(
  data    = main_df,
  y       = "tlx_mental",
  factors = c("Video", "DriverPosition")
)
```

---

check_homogeneity_by_group

Check homogeneity of variances across groups

Description

Check homogeneity of variances across groups

Usage

```
check_homogeneity_by_group(data, x, y)
```

Arguments

data	the data frame
x	the grouping variable (column name as string)
y	the dependent variable (column name as string)

Value

TRUE if Levene's test is non-significant (p >= .05), FALSE otherwise. The Levene test result (columns df1, df2, statistic, p) is attached in the "test" attribute, e.g. for use in a methods section via assumption_methods_text().

---

check_normality_by_group

Check normality for groups

Description

Check normality for groups

Usage

```
check_normality_by_group(data, x, y)
```

Arguments

data	the data frame
x	the x column
y	the y column

Value

TRUE if all groups are normal, FALSE otherwise. The per-group Shapiro-Wilk statistics are attached as a data frame in the "tests" attribute (columns: group, W, p_value), e.g. for use in a methods section via assumption_methods_text(). For groups with more than 5000 non-missing values, Shapiro-Wilk is computed on a random sample of 5000 observations (a warning is emitted); the returned value still reflects that sampled test. Because the sample is drawn randomly, results for such large groups are not reproducible unless a seed is set beforehand.

`check_questionnaire` *Show how a questionnaire will be scored, before scoring it*

Description

Prints the mapping `score_questionnaire()` would use – which column supplies which item, which subscale it loads on, whether it is reverse-coded – along with the observed range of each column and the instrument's scoring notes. Run it once per study: it is the cheapest available check against the failure mode in which a shifted survey export produces perfectly plausible, wrong scores.

Usage

```
check_questionnaire(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
  reverse_items = NULL
)
```

Arguments

`data`	A data frame with one row per respondent (or per respondent and condition) and one column per item.
`instrument`	Instrument key, e.g. `"sus"`, `"nasa_tlx"`, `"ueq_s"`, `"tia"`, `"ssq"`. See `list_questionnaires()`.
`items`	Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (`c(mental = "tlx_md", effort = "tlx_ef", ...)`), which is order-proof and the safer choice for a survey export you did not lay out yourself.
`prefix`	Optional column-name prefix selecting the item columns, e.g. `"sus_"`. Matching columns are sorted numerically, so `sus_2` comes before `sus_10`. Must select exactly as many columns as the instrument has items.
`scale`	Optional two-element vector giving the response range your survey used, e.g. `c(1, 21)` for the 21-point NASA-TLX sheet or `c(0, 4)` for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale.
`reverse_items`	Optional items to *toggle* the reverse-coding of, given as item numbers or item codes. Naming an item the scoring key already reverses un-reverses it, which is what an export that stores a pair the other way round needs.

Value

Invisibly, a data frame with one row per item: the item number and code, the column it maps to, its subscale, whether it is reverse-coded, and the observed minimum, maximum and number of missing values of that column.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself.

See Also

`score_questionnaire()`, `questionnaire_items()`

Examples

```
set.seed(1)
d <- as.data.frame(matrix(sample(1:5, 10 * 4, TRUE), nrow = 4))
names(d) <- paste0("sus_", 1:10)
check_questionnaire(d, "sus", prefix = "sus_")
```

`cite_methods` *Citations and methods boilerplate for the analyses used*

Description

Prints a ready-made methods phrase plus the BibTeX entries for the R packages behind the requested analysis methods, so a manuscript's methods section and bibliography can be filled in one step.

Usage

```
cite_methods(methods = c("ggstatsplot", "effectsize"), bibtex = TRUE)
```

Arguments

`methods`	Character vector of analysis methods to cite. Any of `"art"` (Aligned Rank Transform via ARTool), `"dunn"` (Dunn's test via FSA), `"nparld"` (nparLD), `"ggstatsplot"`, `"effectsize"`, and `"colleyrstats"` (this package).
`bibtex`	whether to include the BibTeX entries. Default `TRUE`.

Value

Invisibly returns the generated lines as a character vector; the text is also emitted via `message()`. Methods whose package is not installed are skipped with a message.

Examples

```
cite_methods("ggstatsplot", bibtex = FALSE)
```

`classify_outcome` *Classify the measurement scale of an outcome variable*

Description

Decides how a dependent variable should be modelled by inspecting its type and distribution of values. The measurement scale is the first branch of a principled model choice: it dictates the *family* (Gaussian, binomial, Poisson, cumulative-link) before any distributional assumption is checked.

Usage

```
classify_outcome(y, ordinal_max_levels = 7L)
```

Arguments

`y`	The outcome vector.
`ordinal_max_levels`	Integer. Integer-valued numerics with at most this many distinct values are treated as ordinal (Likert-like). Default 7.

Details

The rules are deliberately simple and transparent:

- ordered factor → `"ordinal"`;
- logical, a two-level factor, or a numeric/character with exactly two distinct values → `"binary"`;
- unordered factor/character with more than two levels → `"nominal"`;
- integer-valued numeric with at most `ordinal_max_levels` distinct values (a Likert-type item) → `"ordinal"`;
- non-negative integer-valued numeric with more distinct values → `"count"`;
- any other numeric → `"continuous"`.

The heuristics can never be perfect (a 1–7 Likert item and a small count are genuinely ambiguous); pass an explicit `outcome_type` to `recommend_test()` when you want to override them.

Value

A single string, one of `"continuous"`, `"ordinal"`, `"binary"`, `"count"`, or `"nominal"`.

Examples

```
classify_outcome(rnorm(50)) # "continuous"
classify_outcome(factor(sample(1:5, 50, TRUE), ordered = TRUE)) # "ordinal"
classify_outcome(sample(0:1, 50, TRUE)) # "binary"
classify_outcome(rpois(50, 3)) # "count"
```

---

colleyRstats_setup *Configure Global R Environment for colleyRstats*

Description

Sets the ggplot2 theme, and on request the `conflicted` preferences, to match the standards used in the colleyRstats workflow.

Usage

```
colleyRstats_setup(
  set_options = FALSE,
  set_theme = TRUE,
  base_size = 17,
  set_conflicts = FALSE,
  print_citation = TRUE,
  verbose = TRUE
)
```

Arguments

`set_options`	**[Deprecated]** Ignored. colleyRstats no longer changes global `options()`; passing `TRUE` warns and does nothing. Default is `FALSE`.
`set_theme`	Logical. If `TRUE`, sets the default `ggplot2` theme to `see::theme_lucid` with custom modifications. Default is `TRUE`.
`base_size`	Base font size in points for the theme. Every text element is sized relative to it (see `colley_theme()`), so a figure destined for a 3.33 in column and one destined for a slide can share the same theme and differ only in this number. Default 17 reproduces the sizes used before this argument existed. `save_paper_figure()` overrides it per figure to match the width actually being written.
`set_conflicts`	Logical. If `TRUE`, registers this package's `conflicted` preferences, which favor `dplyr` and other tidyverse packages. Default is `FALSE`. Registering them activates `conflicted`, which takes over `library()` for the rest of the session, so it is opt-in and belongs after your `library()` calls – see Details.
`print_citation`	Logical. If `TRUE`, prints the citation information for this package. Default is `TRUE`.
`verbose`	Logical. If `TRUE`, emit informational messages. Default is `TRUE`.

Details

`set_conflicts` **changes what** `library()` **does, so call it last.** `conflicted::conflict_prefer()` does not merely record a preference: it activates `conflicted`, which attaches a `.conflicts` environment carrying its own `library()` and `require()` shims. Every later attach in the session goes through them, and they forward `quietly` and `verbose` into `base::library()` as unevaluated symbols. A package whose `.onAttach` inspects the calling `library()` frame – meta-packages that attach their own constituents do this to decide whether to print a banner – then evaluates those symbols in a frame that does not bind them, and the attach fails outright:

```
colleyRstats_setup(set_conflicts = TRUE)
library(easystats)
#> Error: .onAttach failed in attachNamespace() for 'easystats':
#>   object 'quietly' not found
```

The two mechanisms do not compose, and `colleyRstats_setup()` cannot know whether more `library()` calls are coming. The preferences are therefore opt-in, and the contract when you opt in is that `colleyRstats_setup()` runs *after* everything else is attached:

```
library(colleyRstats)
library(easystats)
library(dplyr)
colleyRstats_setup(set_conflicts = TRUE)   # last
```

That is also the ordering in which the preferences are worth most: `conflicted` resolves only those names that are ambiguous among the packages attached at the time, so a call made before the rest of the script's `library()` calls has less to work with.

Value

Invisibly returns `NULL`.

Examples

```
# Runs everywhere, no extra packages, no session side effects
colleyRstats::colleyRstats_setup(
  set_options = FALSE,
  set_theme = FALSE,
  set_conflicts = FALSE,
  print_citation = FALSE,
  verbose = FALSE
)


# Full setup (requires suggested packages; changes session defaults)
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("see", quietly = TRUE)) {
  local({
    old_theme <- ggplot2::theme_get()
    on.exit(ggplot2::theme_set(old_theme), add = TRUE)
```

```
    colleyRstats::colleyRstats_setup(
      set_options = FALSE,
      set_conflicts = FALSE,   # the default; see Details
      print_citation = FALSE,
      verbose = TRUE
    )

    ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
      ggplot2::geom_point()
  })
}
```

`colley_theme` *The colleyRstats ggplot2 theme*

Description

`see::theme_lucid()` with the sizing and legend conventions used across colleyRstats figures. Every text element is expressed as a multiple of `base_size` rather than in absolute points, which is what lets the same theme serve a 3.33 in journal column and a full-width figure: only `base_size` changes.

Usage

```
colley_theme(base_size = 17, base_family = "")
```

Arguments

`base_size`	Base font size in points. Default 17, which reproduces the sizes colleyRstats used before this function was factored out.
`base_family`	Base font family, passed to `see::theme_lucid()`.

Details

Sizing text in absolute points is the usual reason a figure comes out unreadable. Point sizes do not shrink when the canvas does, so a theme tuned on a large canvas puts 20 pt axis titles on a 3.33 in figure, where they overrun the panel and the axis labels collide. Choose `base_size` from the size the figure is finally PLACED at – roughly the body-text size of the document, or a point or two below it. `save_paper_figure()` does this for you.

The multipliers are, relative to `base_size`: axis titles 1.15, axis text 1.0, plot title 1.65, subtitle 1.0, legend text 0.9, caption 0.8, strip text 1.3.

Value

A ggplot2 theme object.

Examples

```
if (requireNamespace("see", quietly = TRUE)) {
  ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
    ggplot2::geom_point() +
    colley_theme(base_size = 8)   # sized for a single journal column
}
```

`debug_contr_error` *Debug contrast errors in ANOVA-like models*

Description

Debug contrast errors in ANOVA-like models

Usage

```
debug_contr_error(dat, subset_vec = NULL)
```

Arguments

`dat`	A data frame of predictors.
`subset_vec`	Optional logical or numeric index vector used to subset rows before checks.

Value

A list with two elements:

nlevels Integer vector giving the number of levels for each factor variable in `dat`.

levels List of factor level labels for each factor variable in `dat`.

Examples

```
dat <- data.frame(
  group = factor(rep(letters[1:3], each = 3)),
  score = rnorm(9)
)

debug_contr_error(dat = dat)
```

`define_questionnaire` *Register your own questionnaire*

Description

Adds an instrument to the registry so that `score_questionnaire()`, `check_questionnaire()` and `score_reliability()` handle it exactly like a built-in one. Use it for a lab-specific scale, a translated or shortened form, or a published instrument this package does not ship – and put the call in a project's setup script so every analysis in that project scores it the same way.

Usage

```
define_questionnaire(
  key,
  name,
  scale,
  subscale,
  code = NULL,
  label = NULL,
  reverse = integer(0),
  recode = c("none", "center", "zero_base"),
  total = NULL,
  total_name = "Total",
  reference = NA_character_,
  higher = NA_character_,
  notes = character()
)
```

Arguments

`key`	Short identifier, used as the `instrument` argument.
`name`	Full name of the instrument, shown in messages and listings.
`scale`	Two-element vector giving the response range, e.g. `c(1, 7)`.
`subscale`	Character vector, one entry per item, naming the subscale that item loads on. Use a comma-separated string (`"Nausea,Oculomotor"`) for an item that loads on two.
`code`	Optional short code per item, used to refer to items in `reverse_items` and in a named `items` mapping. Defaults to `item1`, `item2`, ...
`label`	Optional item wording, one entry per item.
`reverse`	Optional item numbers that are reverse-scored.
`recode`	One of `"none"` (default), `"center"` (subtract the midpoint, giving the $-3..+3$ coding of a 7-point semantic differential), or `"zero_base"` (subtract the minimum).

total	How to form an overall score across all items: "mean", "sum", or NULL (default) for none – which is the honest choice for a multidimensional instrument whose authors define no total.
total_name	Column name for that overall score. Default "Total".
reference	Optional citation, shown alongside the instrument.
higher	Optional one-word note on what a high score means, e.g. "better" or "worse".
notes	Optional character vector of scoring notes.

Value

Invisibly, the instrument definition.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run check_questionnaire() once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the "mapping" attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with options(colleyRstats.quiet_questionnaires = TRUE)).

A named items argument (items = c(mental = "tlx_md", ...)) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself.

See Also

list_questionnaires(), score_questionnaire()

Examples

```
define_questionnaire(
  key = "acceptance",
  name = "Van der Laan acceptance scale",
  reference = "Van der Laan, Heino & De Waard (1997), Transp. Res. C 5(1)",
  scale = c(-2, 2),
  subscale = c(
    "Usefulness", "Satisfying", "Usefulness", "Satisfying", "Usefulness",
    "Satisfying", "Usefulness", "Satisfying", "Usefulness"
  ),
  label = c(
    "useful - useless", "pleasant - unpleasant", "bad - good",
    "nice - annoying", "effective - superfluous", "irritating - likeable",
   "assisting - worthless", "undesirable - desirable", "raising alertness - sleep-inducing"
  ),
```

```
  reverse = c(1, 2, 4, 5, 7, 9),
  higher = "better"
)
list_questionnaires()[1, ]
```

`define_result_macro` *Define a named LaTeX macro for a single result (single source of truth)*

Description

Emits `\newcommand{\<name>}{<value>}` so you can write `\<name>` in your prose and have it always reflect the latest analysis – re-run the R code and the number updates everywhere it is referenced, the gold standard for reproducible manuscripts. The `name` is sanitised to a valid letters-only LaTeX command name.

Usage

```
define_result_macro(name, value, path = NULL)
```

Arguments

`name`	A label for the result, e.g. `"tlx_mental_omnibus"` (becomes `\tlxMentalOmnibus`).
`value`	The rendered result string, e.g. `"F(2, 57) = 4.50, p = .02"`. It is inserted verbatim (already-formatted LaTeX), not escaped.
`path`	Optional `.tex` path. When supplied and it already exists, the definition is appended (so many results can accumulate in one file).

Value

Invisibly, a named character scalar: the `\newcommand` line, named by the generated command. Also emitted via `message()`.

Examples

```
define_result_macro("tlx_mental_omnibus", "F(2, 57) = 4.50, p = .02")
```

emit_name_macros *Generate \newcommand stubs for variable/factor names*

Description

The report functions can emit variable and factor-level names as LaTeX commands (e.g. `\Video`) so their typography is controlled centrally. This writes the matching `\newcommand` definitions so those commands are never undefined – the classic "Undefined control sequence" that stops an Overleaf build. Only names that are valid LaTeX command names (letters only) get a macro; others are reported as skipped (the reporters emit those as escaped plain text instead).

Usage

```
emit_name_macros(vars, path = NULL, labels = NULL)
```

Arguments

`vars`	Character vector of variable/level names (e.g. the columns you pass as `iv`/`dv`), or a named character vector / list mapping a name to the display label it should expand to.
`path`	Optional `.tex`/`.sty` path to write the definitions to.
`labels`	Optional named character vector mapping a name to its display label (overrides names taken from `vars`).

Value

Invisibly, the `\newcommand` lines as a character vector; also emitted via `message()`.

Examples

```
emit_name_macros(c("Video", "DriverPosition"))
emit_name_macros(c(tlxMental = "TLX Mental Demand"))
```

emit_overleaf *Bundle an analysis into an Overleaf-ready folder*

Description

Writes everything a manuscript needs into one directory you can drag into Overleaf and compile immediately: a `main.tex` that already `\inputs` the results, one `.tex` per result section, the figures, a `references.bib`, and – unless macros are expanded inline – `colleyRstats.sty`. This is the one-call end of the "R analysis to compiled PDF" pipeline.

Usage

```
emit_overleaf(
  x,
  dir,
  figures = TRUE,
  methods = c("ggstatsplot", "effectsize"),
  title = "Results",
  plain = NULL,
  columns = 1,
  overwrite = FALSE
)
```

Arguments

x	What to emit. Accepts a report_all() result (one section per dependent variable, with figures), an analyze_and_report() result, a named list of sentence vectors, or a single character vector.
dir	Output directory (created if needed).
figures	Whether to save figures for sections that carry a plot. Default TRUE.
methods	Methods to cite (passed to cite_methods()) for references.bib; NULL to skip the bibliography.
title	Title used in the generated main.tex. Default "Results".
plain	Whether to expand the colleyRstats macros to plain LaTeX (so no .sty / \usepackage is needed). Default NULL follows getOption("colleyRstats.macros") (i.e. plain when that is FALSE).
columns	Figure width preset passed to save_paper_figure().
overwrite	Overwrite existing files in dir? Default FALSE.

Value

Invisibly, a list with the paths written (dir, main, results, sections, figures, bib, sty).

Examples

```
out <- report_all(mtcars, dvs = c("mpg", "disp"), iv = "cyl")
emit_overleaf(out, dir = file.path(tempdir(), "paper"), overwrite = TRUE)
```

---

expand_latex_macros *Expand the colleyRstats LaTeX macros to plain standard LaTeX*

Description

The report functions normally emit compact custom macros (\F, \p, \m, \sd, \df, \chisq, \padj, \padjminor, \pminor, \rankbiserial, \effectsize) that require latex_preamble() definitions. This expands them into equivalent plain math (e.g. \F{2}{57}{4.50} becomes $F(2, 57) = 4.50$) so the text compiles in any document with no custom preamble – the "zero-setup Overleaf" path. It is applied automatically by the sink_to/emit_overleaf() writers when options(colleyRstats.macros = FALSE).

Usage

```
expand_latex_macros(x)
```

Arguments

x	A character vector of report text.

Value

The text with the macros expanded to standard LaTeX math.

Examples

```
expand_latex_macros("A significant effect (\\F{2}{57}{4.50}, \\p{0.012}).")
```

---

figure_base_size *Base font size for a figure of a given width*

Description

The rule save_paper_figure() uses to pick a type size from a figure width. Text in a figure should read at roughly the body-text size of the document the figure is placed in; since a figure is usually placed at 100\ means the type size has to follow the physical width. The rule is calibrated so that the two standard widths land on sensible values: 3.33 in (a two-column journal column) gives 7 pt and 7 in (full text width) gives 9 pt, with linear interpolation between and clamping outside.

Usage

```
figure_base_size(width, min_size = 6, max_size = 12)
```

Arguments

width	Figure width in inches.
min_size, max_size	Bounds, so that very small or very large figures still get a usable size. Defaults 6 and 12 points.

Value

A single number: the base font size in points.

Examples

```
figure_base_size(3.33)  # 7
figure_base_size(7)     # 9
```

`fit_recommended` *Fit the model that the data call for, and report it*

Description

Runs `recommend_test()` to choose an analysis, then actually fits it: coerces the outcome and predictors into the classes that model family needs, builds the random-effect term for a repeated-measures design, fits, computes pairwise post-hoc contrasts with the matching machinery, and produces the APA/LaTeX sentence via this package's reporter for that model family.

Usage

```
fit_recommended(
  data,
  outcome,
  predictors = NULL,
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L,
  interaction = TRUE,
  contrasts = TRUE,
  adjust = "holm",
  sink_to = NULL,
  verbose = TRUE
)
```

Arguments

`data`	The data frame.
`outcome`	The dependent variable (column name as string).
`predictors`	Character vector of independent variable column names.
`cluster`	Optional column identifying the subject or cluster for repeated-measures data – the random-effect grouping factor.
`design`	One of `"auto"` (default; clustered when `cluster` is given), `"between"`, or `"within"`.

outcome_type	"auto" (default; use classify_outcome()) or an explicit "continuous", "ordinal", "binary", "count", "nominal" to override the classification. Pass it when the automatic choice is wrong – a 1-7 Likert item and a small count are genuinely ambiguous from the data alone.
ordinal_max_levels	Passed to classify_outcome(). Default 7.
interaction	Logical. With more than one predictor, fit the full factorial model (TRUE, default) or main effects only (FALSE).
contrasts	Logical. Compute pairwise post-hoc contrasts between the levels of the grouping predictors. Default TRUE.
adjust	Multiplicity adjustment for those contrasts, passed to **emmeans** or **ARTool**. Default "holm".
sink_to	Optional path of a .tex file; the methods sentence and the result sentence are written there so a manuscript can \input{} them.
verbose	Logical. If TRUE (default), report every coercion and the model being fitted.

Details

The point is that the test you justify in the methods section and the model you actually ran are produced by the same call, from the same data, and therefore cannot drift apart – the failure mode of a pipeline where recommend_test() prints advice that someone then re-types by hand.

Value

An object of class "colley_fit": a list with recommendation (the recommend_test() object), model (the fitted object), engine, emmeans and contrasts (or NULL), methods and text (manuscript sentences), and sentences (both, in manuscript order). A print method summarises it.

What it can fit

Cumulative link models with and without random effects (**ordinal**), generalized and linear mixed models (**lme4**, **lmerTest**), GLMs, the aligned rank transform (**ARTool**), rank-based repeated measures (**nparLD**), multinomial regression (**nnet**), and the classical ANOVA / Welch / Kruskal-Wallis / Wilcoxon tests. Model packages live in Suggests: a branch you use needs its package installed, and says which one if it is not.

See Also

recommend_test() for the choice alone, analyze_and_report() for the figure-plus-sentence path through **ggstatsplot**.

Examples

```
set.seed(1)
d <- data.frame(
  id = factor(rep(1:20, each = 3)),
  cond = factor(rep(c("A", "B", "C"), times = 20)),
  score = rnorm(60)
```

```
)
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- fit_recommended(d, outcome = "score", predictors = "cond", cluster = "id")
  fit
}
```

`generateEffectPlot` *Function to define a plot, either showing the main or interaction effect in bold.*

Description

Function to define a plot, either showing the main or interaction effect in bold.

Usage

```
generateEffectPlot(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
  numberColors = 6
)

plot_effect(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
```

```
  numberColors = 6
)
```

Arguments

data	the data frame
x	factor shown on the x-axis
y	dependent variable
fillColourGroup	group to color
ytext	label for y-axis
xtext	label for x-axis
legendPos	position for legend
legendHeading	custom heading for legend
shownEffect	either "main" or "interaction"
effectLegend	TRUE: show legend for effect (Default: FALSE)
effectDescription	custom label for effect
xLabelsOverwrite	custom labels for x-axis
useLatexMarkup	use latex font and markup
numberColors	number of colors

Value

a plot

Naming

`plot_effect()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`generateEffectPlot()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)
main_df <- data.frame(
  strategy    = factor(rep(c("A", "B"), each = 20)),
  Emotion     = factor(rep(c("Happy", "Sad"), times = 20)),
  trust_mean  = rnorm(40, mean = 5, sd = 1)
)

generateEffectPlot(
  data = main_df,
```

```
  x = "strategy",
  y = "trust_mean",
  fillColourGroup = "Emotion",
  ytext = "Trust",
  xtext = "Strategy",
  legendPos = c(0.1, 0.23)
)
```

`generateMoboPlot` *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases.

Usage

```
generateMoboPlot(
  data,
  x,
  y,
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)

plot_mobo(
  data,
  x,
  y,
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)
```

Arguments

`data`	A data frame containing the data to be plotted.

`x` A string representing the column name in `data` to be used for the x-axis. Can be either numeric or factor.

`y` A string representing the column name in `data` to be used for the y-axis. This should be a numeric variable.

`fillColourGroup` A string representing the column name in `data` that defines the fill color grouping for the plot. Default is `"ConditionID"`.

`ytext` A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of `y`.

`legendPos` A numeric vector of length 2 specifying the position of the legend inside the plot. Default is `c(0.65, 0.85)`.

`numberSamplingSteps` An integer specifying the number of initial sampling steps before the optimization phase begins. Default is 5.

`labelPosFormulaY` A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are `"top"`, `"center"`, or `"bottom"`. Default is `"top"`.

`verticalLinePosY` A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn.

Value

A `ggplot` object representing the multi-objective optimization plot, ready to be rendered.

Naming

`plot_mobo()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`generateMoboPlot()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y")

# Example with factor x-axis
df <- data.frame(
```

```
  x = factor(rep(1:5, each = 4)),
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y", numberSamplingSteps = 3)
```

`generateMoboPlot2` *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases. Appropriate if you use https://github.com/Pascal-Jansen/Bayesian-Optimization-for-Unity in version 1.1.0 or higher.

Usage

```
generateMoboPlot2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
  fillLabels = NULL,
  annotationTextSize = 5
)

plot_mobo2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
```

```
  fillLabels = NULL,
  annotationTextSize = 5
)
```

Arguments

data	A data frame containing the data to be plotted.
x	A string representing the column name in data to be used for the x-axis. Can be either numeric or factor. Default is "Iteration".
y	A string representing the column name in data to be used for the y-axis. This should be a numeric variable.
phaseCol	the name of the column for the color of the phase (sampling or optimization)
fillColourGroup	A string representing the column name in data that defines the fill color grouping for the plot. Default is "ConditionID".
ytext	A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of y.
legendPos	A numeric vector of length 2 specifying the position of the legend inside the plot. Default is c(0.65, 0.85).
labelPosFormulaY	A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are "top", "center", or "bottom". Default is "top".
labelPosFormulaX	A string specifying the position of the polynomial equation label in the plot. Acceptable values are "left", "center", or "right". Default is "left".
horizontalLinePosY	A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn. Default is 0.75
horizontalLineDistToText	A numeric value of the y-coordinate where the "sampling" and "optimization" text should be drawn below the line. Default is 0.3
fillLabels	An optional named character vector mapping raw factor levels to display labels for the fill/colour legend (e.g. c("value_only" = "Value Only", "llm_only" = "LLM Only")). If NULL (default), the original factor levels are used as-is.
annotationTextSize	numeric. The font size for embedded text annotations inside the plot (e.g., "Sampling", "Optimization" labels, and the regression equations). Default is 5.0.

Value

A ggplot object representing the multi-objective optimization plot, ready to be rendered.

Naming

plot_mobo2() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

`generateMoboPlot2()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10),
  Phase = rep(c("Sampling", "Optimization"), 10)
)
generateMoboPlot2(data = df, x = "x", y = "y")
```

`ggbetweenstatsWithPriorNormalityCheck`

Check the data's distribution. If non-normal, take the non-parametric variant of ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggbetweenstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)

plot_between_stats(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
```

```
  plotType = "boxviolin"
)
```

Arguments

data	the data frame
x	the independent variable, most likely "ConditionID"
y	the dependent variable under investigation
ylab	label to be shown for the dependent variable
xlabels	labels to be used for the x-axis
showPairwiseComp	whether to show pairwise comparisons, TRUE as default
plotType	either "box", "violin", or "boxviolin" (default)

Value

A ggplot object produced by ggstatsplot::ggbetweenstats, which can be printed or further modified with +.

Naming

plot_between_stats() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggbetweenstatsWithPriorNormalityCheck() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID",
  y = "tlx_mental", ylab = "Mental Demand",
  xlabels = labels_xlab,
```

```
  showPairwiseComp = TRUE
)
```

`ggbetweenstatsWithPriorNormalityCheckAsterisk`

Check the data's distribution. If non-normal, take the non-parametric variant of ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin"
)

plot_between_stats_asterisk(data, x, y, ylab, xlabels, plotType = "boxviolin")
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggbetweenstats` with additional significance annotations, which can be printed or modified.

Naming

`plot_between_stats_asterisk()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`ggbetweenstatsWithPriorNormalityCheckAsterisk()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
  x = "CondID", y = "tlx_mental", ylab = "Mental Demand", xlabels = labels_xlab
)
```

---

ggwithinstatsWithPriorNormalityCheck

Check the data's distribution. If non-normal, take the non-parametric variant of ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggwithinstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
```

```
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)

plot_within_stats(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)
```

Arguments

data	the data frame
x	the independent variable, most likely "ConditionID"
y	the dependent variable under investigation
ylab	label to be shown for the dependent variable
xlabels	labels to be used for the x-axis
showPairwiseComp	whether to show pairwise comparisons, TRUE as default
plotType	either "box", "violin", or "boxviolin" (default)

Value

A ggplot object produced by ggstatsplot::ggwithinstats with additional significance annotations, which can be printed or modified.

Naming

plot_within_stats() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggwithinstatsWithPriorNormalityCheck() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
#'   set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
```

```
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand",
  xlabels = labels_xlab,
  showPairwiseComp = TRUE
)
```

---

ggwithinstatsWithPriorNormalityCheckAsterisk

Check the data's distribution. If non-normal, take the non-parametric variant of ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID". Add Asterisks instead of p-values.*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID". Add Asterisks instead of p-values.

Usage

```
ggwithinstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin"
)

plot_within_stats_asterisk(data, x, y, ylab, xlabels, plotType = "boxviolin")
```

Arguments

data	the data frame
x	the independent variable, most likely "ConditionID"
y	the dependent variable under investigation
ylab	label to be shown for the dependent variable
xlabels	labels to be used for the x-axis
plotType	either "box", "violin", or "boxviolin" (default)

Value

A ggplot object produced by ggstatsplot::ggwithinstats with additional significance annotations, which can be printed or modified.

Naming

plot_within_stats_asterisk() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

ggwithinstatsWithPriorNormalityCheckAsterisk() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand", xlabels = labels_xlab
)
```

---

latexify_report *Transform text from* report::report() *into LaTeX-friendly output.*

Description

This function transforms the text output from report::report() by performing several substitutions to prepare the text for LaTeX typesetting. In particular, it replaces instances of R2, %, and ~ with the corresponding LaTeX code. Additionally, it provides options to:

- Omit bullet items marked as "non-significant" (when only_sig = TRUE).
- Remove a concluding note about standardized parameters (when remove_std = TRUE).
- Wrap bullet items in a LaTeX itemize environment or leave them as plain text (controlled by itemize).

Usage

```
latexify_report(
  x,
  print_result = TRUE,
  only_sig = FALSE,
  remove_std = FALSE,
  itemize = TRUE
)
```

Arguments

x	Character vector or a single string containing the report text.
print_result	Logical. If TRUE (default), the formatted text is printed to the console.
only_sig	Logical. If TRUE, bullet items containing "non-significant" are omitted. Default is FALSE.
remove_std	Logical. If TRUE, the final standardized parameters note is removed. Default is FALSE.
itemize	Logical. If TRUE (default), bullet items are wrapped in a LaTeX itemize environment; otherwise the bullet markers are simply removed.

Value

A single string with the LaTeX-friendly formatted report text.

Examples

```
if (requireNamespace("report", quietly = TRUE)) {
  # Simple linear model on the iris dataset
  model <- stats::lm(
    Sepal.Length ~ Sepal.Width + Petal.Length,
    data = datasets::iris
  )

  # Format the report output, showing only significant items, removing the
  # standard note, and wrapping bullet items in an itemize environment.
  report_text <- try(report::report(model), silent = TRUE)
  if (!inherits(report_text, "try-error")) {
    latexify_report(
      report_text,
      only_sig = TRUE,
      remove_std = TRUE,
      itemize = TRUE
    )
  }
}
```

latex_escape *Escape LaTeX special characters in plain text*

Description

Makes an arbitrary string safe to drop into a LaTeX document by escaping the characters that would otherwise be interpreted as markup (\ { } $ & # _ % ~ ^ < >). Use it on variable names, factor-level labels, captions – anything user-supplied that reaches the `.tex`. This is what prevents a dependent variable called `tlx_mental` from producing an un-compilable `tlx_mental` (a subscript error) in Overleaf.

Usage

```
latex_escape(x)
```

Arguments

x	A character vector (or something coercible to one).

Value

A character vector with LaTeX specials escaped; `NA` is preserved.

Examples

```
latex_escape("tlx_mental")
latex_escape("cost (%) & margin")
```

latex_preamble *LaTeX preamble required by the report functions*

Description

All report functions emit LaTeX text that relies on a small set of custom commands. This helper prints the complete set, ready to paste into a manuscript preamble, or writes it to a file that can be included with `\input{}` (or renamed to `.sty` and loaded via `\usepackage`).

Usage

```
latex_preamble(path = NULL)
```

Arguments

path	Optional path of a `.tex` file to write the definitions to. If the path ends in `.sty`, a `\ProvidesPackage` header is added so it can be uploaded to Overleaf and loaded with `\usepackage{colleyRstats}` (see also `use_colleyrstats_sty()`).

Value

Invisibly returns the macro definitions as a character vector; the text is also emitted via `message()`.

Examples

```
latex_preamble()
```

`list_questionnaires` *List the questionnaires this package can score*

Description

List the questionnaires this package can score

Usage

```
list_questionnaires()
```

Value

A data frame with one row per instrument: its key (the value to pass as `instrument`), full name, number of items, number of subscales, the response range it assumes, the direction a high score means, and its reference.

See Also

`questionnaire_items()`, `score_questionnaire()`, `define_questionnaire()`

Examples

```
list_questionnaires()
```

`na.zero` *Replace NA values with zero*

Description

Replace NA values with zero

Usage

```
na.zero(x)
```

Arguments

x	A vector.

Value

A vector with NAs replaced by zeros.

Examples

```
na.zero(c(NA, 1, NA, 2))
```

`normalize` *This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].*

Description

This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].

Usage

```
normalize(x_vector, old_min, old_max, new_min, new_max)
```

Arguments

`x_vector`	A numeric vector that you want to normalize.
`old_min`	The minimum value in the original scale of the data.
`old_max`	The maximum value in the original scale of the data.
`new_min`	The minimum value in the new scale to which you want to normalize the data.
`new_max`	The maximum value in the new scale to which you want to normalize the data.

Value

A numeric vector with the normalized values.

Examples

```
normalize(c(1, 2, 3, 4, 5), 1, 5, 0, 1)
```

not_empty *Ensure input is not empty*

Description

Stops execution if x is NULL, empty, or contains only NAs.

Usage

```
not_empty(x, msg = NULL)
```

Arguments

x	The object to check
msg	The error message to display. The default names the offending argument, e.g. "`data` must not be empty.".

Value

Invisible TRUE if valid.

not_in *Negate %in% membership*

Description

Negate %in% membership

Usage

```
not_in(x, y)

x %!in% y
```

Arguments

x	Vector of values to test.
y	Vector of values to match against.

Value

Logical vector indicating non-membership.

n_fun *Build a median/size label for plot annotations*

Description

Build a median/size label for plot annotations

Usage

```
n_fun(x)
```

Arguments

x	A numeric vector.

Value

A data frame with the median and label.

pathPrep *Convert Windows paths to R-friendly format*

Description

Convert Windows paths to R-friendly format

Usage

```
pathPrep(path = "clipboard", read_fn = NULL, write_fn = NULL)
```

Arguments

path	Path to convert or the string "clipboard" to read from the clipboard.
read_fn	Optional custom function to read from the clipboard.
write_fn	Optional custom function to write to the clipboard.

Value

A normalized path string.

`questionnaire_items` *The items of one questionnaire*

Description

The items of one questionnaire

Usage

```
questionnaire_items(instrument, notes = TRUE)
```

Arguments

`instrument`	Instrument key, e.g. `"tia"`. See `list_questionnaires()`.
`notes`	Logical. If `TRUE` (default), also emit the instrument's scoring notes – how the overall score is formed, what its range means, and where its published form is known to vary between administrations.

Value

Invisibly, a data frame with one row per item: number, code, wording, subscale, and whether the scoring key reverses it.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run `check_questionnaire()` once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the `"mapping"` attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with `options(colleyRstats.quiet_questionnaires = TRUE)`).

A named `items` argument (`items = c(mental = "tlx_md", ...)`) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself.

See Also

`check_questionnaire()`, `score_questionnaire()`

Examples

```
questionnaire_items("ueq_s")
```

`recommend_test` *Recommend a principled analysis for one outcome*

Description

Works out, from the data alone, which statistical model is appropriate for a given outcome and set of predictors, and – crucially – *why*. The decision follows a transparent three-question tree:

1. **What is the outcome's measurement scale?** (via `classify_outcome()`: continuous, ordinal, binary, count, nominal.) This fixes the model family.
2. **Are the observations independent or clustered?** A repeated-measures / clustered design (a `cluster` id is supplied, or `design = "within"`) needs random effects, i.e. a *mixed* model.
3. **For a continuous outcome, do the parametric assumptions hold?** Group-wise normality (Shapiro–Wilk) and, for between-subjects designs, homogeneity of variance (Levene) decide between a parametric test and a non-parametric / rank-based alternative.

Usage

```
recommend_test(
  data,
  outcome,
  predictors = NULL,
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L
)

recommend_analysis(
  data,
  outcome,
  predictors = NULL,
  cluster = NULL,
  design = c("auto", "between", "within"),
 outcome_type = c("auto", "continuous", "ordinal", "binary", "count", "nominal"),
  ordinal_max_levels = 7L
)
```

Arguments

`data`	The data frame.
`outcome`	The dependent variable (column name as string).
`predictors`	Optional character vector of predictor (independent variable) column names.
`cluster`	Optional column name identifying the subject/cluster for repeated-measures or otherwise non-independent data (the random-effect grouping factor).

design One of "auto" (default; clustered when cluster is given), "between", or "within".

outcome_type One of "auto" (default; use classify_outcome()) or an explicit "continuous", "ordinal", "binary", "count", "nominal" to override the automatic classification.

ordinal_max_levels
Passed to classify_outcome(). Default 7.

Details

The recommendation therefore ranges over ordinary ANOVA / t-tests, rank-based methods (Kruskal–Wallis + Dunn, Wilcoxon, the Aligned Rank Transform, nparLD), generalized linear models, cumulative link models, and their mixed-model counterparts – linear mixed models (LMM), generalized linear mixed models (GLMM, lme4/glmmTMB) and cumulative link mixed models (CLMM, ordinal).

Value

An object of class "colley_recommendation" (a list) with components including outcome_type, clustered, family, recommendation (human-readable label), model_function (the R function to call, e.g. "ordinal::clmm"), reporter (the matching colleyRstats reporter), fit_call (a ready-to-edit call as a string), alternatives, rationale, and methods_text (an APA-style sentence). A print method summarises it.

Naming

recommend_test() is the canonical name. recommend_analysis() is an alias for the same function object, kept for backward compatibility.

Examples

```
set.seed(1)
d <- data.frame(
  id    = factor(rep(1:20, each = 3)),
  cond  = factor(rep(c("A", "B", "C"), times = 20)),
  score = rnorm(60),
  rating = factor(sample(1:5, 60, TRUE), ordered = TRUE)
)
# Ordinal outcome measured repeatedly within subject -> CLMM
recommend_test(d, outcome = "rating", predictors = "cond", cluster = "id")
# Continuous, between-subjects -> ANOVA or its rank-based fallback
recommend_test(d, outcome = "score", predictors = "cond")
```

---

remove_outliers_REI *Flag suspicious survey responses via the Response Entropy Index (REI)*

Description

This function takes a data frame, optional header information, variables to consider, and a range for a Likert scale. It then calculates the Response Entropy Index (REI) and flags suspicious entries based on percentiles. Note that no rows are removed; entries are only flagged via the `Suspicious` column.

Usage

```
remove_outliers_REI(df, header = FALSE, variables = "", range = c(1, 5))
```

Arguments

`df`	Data frame containing the data.
`header`	Logical indicating if the data frame has a header. Defaults to FALSE.
`variables`	Which variables to consider: either a single character string with names separated by commas (`"var1,var2"`) or a character vector (`c("var1", "var2")`).
`range`	Numeric vector of length 2 specifying the range of the Likert scale (used to sanity-check the responses). Defaults to c(1, 5).

Details

Missing responses are ignored when tallying answers. Responses outside the declared Likert `range` trigger a warning (they often indicate mis-coded data) but are still included in the REI computation.

For more information on the REI method, refer to: Response Entropy Index Method

Value

A data frame with calculated REI, percentile, and a 'Suspicious' flag.

Examples

```
df <- data.frame(var1 = c(1, 2, 3), var2 = c(2, 3, 4))
result <- remove_outliers_REI(df, TRUE, "var1,var2", c(1, 5))
```

`replace_values` *Replace values across a data frame*

Description

Replace all occurrences of given values in all columns of a data frame. Factor levels are preserved (and extended by the replacement values), and numeric/logical columns are only touched where a value actually matches, so unrelated entries keep their exact binary representation.

Usage

```
replace_values(data, to_replace, replace_with)
```

Arguments

data	The input data frame to be modified.
to_replace	A vector of values to be replaced within the data frame. This must be the same length as replace_with.
replace_with	A vector of corresponding replacement values. This must be the same length as to_replace.

Value

Modified data frame with specified values replaced.

Examples

```
data <- data.frame(
  q1 = c("neg2", "neg1", "0"),
  q2 = c("1", "neg2", "neg1")
)

replace_values(
  data,
  to_replace = c("neg2", "neg1"),
  replace_with = c("-2", "-1")
)
```

---

reportART

Generate the Latex-text based on the ARTool (see https://github.com/mjskay/ARTool). The ART result must be piped into an anova(). Only significant main and interaction effects are reported. P-values are rounded for the third digit. Attention: Effect sizes are not calculated! Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).

Description

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: \newcommand{\F}[3]{$F({#1},{#2})={#3}$} \newcommand{\p}{\textit{p=}} \newcommand{\pminor}{\text

Usage

```
reportART(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

```
report_art(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

model	the model of the art
dv	the name of the dependent variable that should be reported
write_to_clipboard	whether to write to the clipboard
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

report_art() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportART() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE)) {
  set.seed(123)

  main_df <- data.frame(
    tlx_mental = stats::rnorm(80),
    Video      = factor(rep(c("A", "B"), each = 40)),
    gesture    = factor(rep(c("G1", "G2"), times = 40)),
    eHMI       = factor(rep(c("On", "Off"), times = 40)),
    UserID     = factor(rep(1:20, each = 4))
  )

  art_model <- ARTool::art(
    tlx_mental ~ Video * gesture * eHMI +
      Error(UserID / (gesture * eHMI)),
    data = main_df
  )

  model_anova <- stats::anova(art_model)
  reportART(model_anova, dv = "mental demand")
}
```

`reportArtCon` *Report significant ART contrasts (art.con) as LaTeX text*

Description

Companion to `reportDunnTest()` for aligned-rank-transform (ART) models. It extracts the significant pairwise comparisons produced by `ARTool::art.con()` (an **emmeans** contrast grid), computes the mean and standard deviation of the groups involved from the raw data, and prints LaTeX-formatted sentences.

Usage

```
reportArtCon(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  sink_to = NULL
)

report_art_con(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  sink_to = NULL
)
```

Arguments

`ac`	the contrast object returned by `ARTool::art.con()` (or its `summary()`)
`data`	the raw data frame used to fit the model
`iv`	independent variable (the contrasted factor)
`dv`	dependent variable
`paired`	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to `FALSE`. When `TRUE`, `id` is required.
`id`	the subject/pairing column, used only when `paired = TRUE`. Replicate trials per subject and condition are averaged before pairing.
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

The p-values are taken as-is from the contrast object, i.e. they are already adjusted by whatever `adjust` was passed to `art.con()` (e.g. `"holm"`). The effect size is the rank-biserial correlation computed from the raw data. ART is most often used for within-subjects designs; pass `paired = TRUE` together with `id` (the subject column) to obtain the paired rank-biserial effect size.

Attention: `ac` must be a pairwise contrast over a single factor `iv` (e.g. `art.con(model, ~ interaction_mode, adjust = "holm")`).

Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}} \newcommand{\padj}{\textit{p$_`
`\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_art_con()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportArtCon()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
  reportArtCon(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportArtConTable` *Report ART contrasts (art.con) as a LaTeX table. Customizable with sensible defaults. Companion to* `reportDunnTestTable()`.

Description

Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}} \newcommand{\padj}{\textit{p$_`
`\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Usage

```
reportArtConTable(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)

report_art_con_table(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)
```

Arguments

ac	the contrast object returned by ARTool::art.con() (or its summary())
data	the raw data frame used to fit the model
iv	independent variable (the contrasted factor)
dv	dependent variable
paired	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to FALSE. When TRUE, id is required.
id	the subject/pairing column, used only when paired = TRUE. Replicate trials per subject and condition are averaged before pairing.
orderByP	whether to order by the p value

numberDigitsForPValue
: the number of digits to show

latexSize
: which size for the text

orderText
: whether to order the comparisons alphabetically; ignored when orderByP = TRUE

style
: table rule style: "hline" (default) or "booktabs" (\toprule/\midrule/\bottomrule; needs \usepackage{booktabs}).

sink_to
: optional path of a .tex file to write the table to, so a manuscript can \input{} it

Value

Invisibly returns the rendered LaTeX table as a string (or NULL when xtable is unavailable); the table is also printed.

Naming

report_art_con_table() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

reportArtConTable() **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
 reportArtConTable(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportCLMM` *Report a cumulative link (mixed) model in LaTeX/APA style*

Description

Reporter for ordinal proportional-odds models fitted with **ordinal**: cumulative link mixed models (`ordinal::clmm`) and their fixed-effects counterpart (`ordinal::clm`). Each location (predictor) effect is reported as an odds ratio – the multiplicative change in the odds of being in a higher outcome category for a one-unit increase in the predictor – with its confidence interval, z statistic and p-value. The threshold (cut-point) coefficients are omitted, as is conventional.

Usage

```
reportCLMM(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_clmm(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	A fitted `ordinal::clmm` or `ordinal::clm` model.
`dv`	Name of the (ordinal) dependent variable, used in the sentence text.
`exponentiate`	`"auto"` (default; report odds ratios) or `TRUE`/`FALSE` to force it. `FALSE` reports raw log-odds.
`conf_level`	Confidence level for the intervals. Default 0.95.
`write_to_clipboard`	Whether to copy the sentences to the clipboard.
`sink_to`	Optional path of a `.tex` file to write the sentences to.

Details

The threshold (cut-point) parameters are never reported, so unlike `reportGLMM()` this reporter has no `include_intercept` argument.

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_clmm()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportCLMM()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("ordinal", quietly = TRUE) &&
  requireNamespace("parameters", quietly = TRUE)) {
  m <- ordinal::clmm(rating ~ temp + contact + (1 | judge), data = ordinal::wine)
  reportCLMM(m, dv = "wine rating")
}
```

`reportDunnTest` *Report dunnTest as text. Required commands in LaTeX:* `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb}` = `#1$}`

Description

Report dunnTest as text. Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb}` = `#1$}`

Usage

```
reportDunnTest(d, data, iv = "testiv", dv = "testdv", sink_to = NULL)

report_dunn_test(d, data, iv = "testiv", dv = "testdv", sink_to = NULL)
```

Arguments

`d`	the dunn test object
`data`	the data frame
`iv`	independent variable
`dv`	dependent variable
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_dunn_test()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportDunnTest()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)

  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTest(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

`reportDunnTestTable` *report Dunn test as a table. Customizable with sensible defaults. Required commands in LaTeX:* `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Description

report Dunn test as a table. Customizable with sensible defaults. Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Usage

```
reportDunnTestTable(
  d = NULL,
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)

report_dunn_test_table(
  d = NULL,
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  style = c("hline", "booktabs"),
  sink_to = NULL
)
```

Arguments

d	the dunn test object
data	the data frame
iv	independent variable
dv	dependent variable
orderByP	whether to order by the p value
numberDigitsForPValue	the number of digits to show
latexSize	which size for the text
orderText	whether to order the comparisons alphabetically; ignored when orderByP = TRUE
style	table rule style: "hline" (default, classic \hline rules) or "booktabs" (journal-standard \toprule/\midrule/\bottomrule; needs \usepackage{booktabs}).
sink_to	optional path of a .tex file to write the table to, so a manuscript can \input{} it

Value

Invisibly returns the rendered LaTeX table as a string (or `NULL` when xtable is unavailable); the table is also printed.

Naming

`report_dunn_test_table()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportDunnTestTable()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)

  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTestTable(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

---

reportggstatsplot *Report statistical details for ggstatsplot.*

Description

Report statistical details for ggstatsplot.

Usage

```
reportggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
```

```
  sink_to = NULL
)

report_ggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

p	the object returned by ggwithinstats or ggbetweenstats
iv	the independent variable
dv	the dependent variable
write_to_clipboard	whether to write to the clipboard
sink_to	optional path of a .tex file to write the sentence to, so a manuscript can \input{} it

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_ggstatsplot()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportggstatsplot()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot
plt <- ggbetweenstats(mtcars, am, mpg)

reportggstatsplot(plt, iv = "am", dv = "mpg")
```

`reportggstatsplotPostHoc` *Report significant post-hoc pairwise comparisons*

Description

This function extracts significant pairwise comparisons from a `ggstatsplot` object, calculates the mean and standard deviation for the groups involved using the raw data, and prints LaTeX-formatted sentences reporting the results.

Usage

```
reportggstatsplotPostHoc(
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL
)

report_ggstatsplot_posthoc(
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL
)
```

Arguments

`data`	A data frame containing the raw data used to generate the plot.
`p`	A `ggstatsplot` object (e.g., returned by `ggbetweenstats`) containing the pairwise comparison statistics.
`iv`	Character string. The column name of the independent variable (grouping variable).
`dv`	Character string. The column name of the dependent variable.
`label_mappings`	Optional named list or vector. Used to rename factor levels in the output text (e.g., `list("old_name" = "New Label")`).
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

LaTeX Requirements

To easily copy and paste the results to your manuscript, the following commands (or similar) must be defined in your LaTeX preamble, as the function outputs commands taking arguments (e.g., `\m{value}`):

```
\newcommand{\m}[1]{\textit{M}=#1}
\newcommand{\sd}[1]{\textit{SD}=#1}
\newcommand{\padj}[1]{$p_{adj}=#1$}
\newcommand{\padjminor}[1]{$p_{adj}<#1$}
```

Naming

`report_ggstatsplot_posthoc()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportggstatsplotPostHoc()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot
plt <- ggbetweenstats(mtcars, am, mpg)

# Report stats
reportggstatsplotPostHoc(
  data = mtcars,
  p = plt,
  iv = "am",
  dv = "mpg",
  label_mappings = list("0" = "Automatic", "1" = "Manual")
)
```

`reportGLMM` *Report a (generalized) linear mixed model in LaTeX/APA style*

Description

Turns a fitted mixed model into ready-to-paste manuscript sentences, one per fixed-effect term, with the coefficient (or odds/incidence-rate ratio for non-Gaussian families), its confidence interval, the test statistic, and the p-value. Works with linear mixed models (`lme4::lmer`), generalized linear mixed models (`lme4::glmer`, `glmmTMB::glmmTMB`) and, for convenience, ordinary `lm`/`glm` fits. Coefficients are exponentiated automatically for binomial (odds ratios) and Poisson/negative-binomial (incidence-rate ratios) families.

Usage

```
reportGLMM(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  include_intercept = FALSE,
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_glmm(
  model,
  dv = "Testdependentvariable",
  exponentiate = "auto",
  include_intercept = FALSE,
  conf_level = 0.95,
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

model	A fitted model (lmer, glmer, glmmTMB, lm, or glm).
dv	Name of the dependent variable, used in the sentence text.
exponentiate	"auto" (default; exponentiate for binomial and count families), or TRUE/FALSE to force it.
include_intercept	Whether to also report the intercept. Default FALSE.
conf_level	Confidence level for the intervals. Default 0.95.
write_to_clipboard	Whether to copy the sentences to the clipboard.
sink_to	Optional path of a .tex file to write the sentences to, so a manuscript can \input{} them.

Details

The reported statistics rely on the **parameters** package. The LaTeX output uses the \p/\pminor macros from latex_preamble().

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Naming

report_glmm() is the spelling used throughout the documentation and the one to prefer in new code: the report_* / plot_* / check_* prefixes make the API discoverable through autocomplete.

`reportGLMM()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("lme4", quietly = TRUE) &&
  requireNamespace("parameters", quietly = TRUE)) {
  m <- lme4::lmer(Reaction ~ Days + (1 | Subject), data = lme4::sleepstudy)
  reportGLMM(m, dv = "reaction time")
}
```

`reportMeanAndSD` *Report the mean and standard deviation of a dependent variable for all levels of an independent variable rounded to the 2nd digit.*

Description

#' To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\m}{\textit{M=}} \newcommand{\sd}{\textit{SD=}}`

Usage

```
reportMeanAndSD(data, iv = "testiv", dv = "testdv", sink_to = NULL)

report_mean_sd(data, iv = "testiv", dv = "testdv", sink_to = NULL)
```

Arguments

`data`	the data frame
`iv`	the independent variable
`dv`	the dependent variable
`sink_to`	optional path of a `.tex` file to write the lines to, so a manuscript can `\input{}` them

Value

Invisibly returns the formatted lines as a character vector; the text is also emitted via `message()`.

Naming

`report_mean_sd()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportMeanAndSD()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
example_data <- data.frame(Condition = rep(c("A", "B", "C"),
each = 10), TLX1 = stats::rnorm(30))

reportMeanAndSD(example_data, iv = "Condition", dv = "TLX1")
```

`reportNparLD` *Report the model produced by nparLD. The model provided must be the model generated by the command 'nparLD'* nparLD *(see* https://CRAN.R-project.org/package=nparLD*).*

Description

#' Only significant main and interaction effects are reported. P-values are rounded for the third digit and relative treatment effects (RTE) are included when available. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).

Usage

```
reportNparLD(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_nparld(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	the model
`dv`	the dependent variable
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

#' To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}{\textit{F=}}` `\newcommand{\df}{\textit{df=}}` `\newcommand{\p}{\textit{p=}}` `\newcommand{\pminor}{\textit{p$<$}}`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Naming

`report_nparld()` is the spelling used throughout the documentation and the one to prefer in new code: the `report_*` / `plot_*` / `check_*` prefixes make the API discoverable through autocomplete.

`reportNparLD()` **[Superseded]** is the original name. Both names refer to the same function object, so they are entirely interchangeable; the original remains fully supported and is not scheduled for removal, and existing scripts keep working unchanged.

Examples

```
if (requireNamespace("nparLD", quietly = TRUE)) {
  # Small toy data set for nparLD
  set.seed(123)
  example_data <- data.frame(
    Subject = factor(rep(1:10, each = 3)),
    Time    = factor(rep(c("T1", "T2", "T3"), times = 10)),
    # a rising trend over time, so there is something to report
    TLX1    = rep(c(45, 52, 61), times = 10) + stats::rnorm(30, sd = 4)
  )

  # Fit nparLD model
  model <- nparLD::nparLD(TLX1 ~ Time, data = example_data, subject = "Subject")

  # Report the nparLD result
  reportNparLD(model, dv = "TLX1")
}
```

`reportNPAV` *Generate the Latex-text based on the NPAV by Lüpsen (see https://www.uni-koeln.de/~{}luepsen/R/). Only significant main and interaction effects are reported. P-values are rounded for the third digit and partial eta squared values are provided when possible. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).*

Description

Deprecated: `reportNPAV()` will be removed in a future release. Use `reportART()` with ARTool instead.

Usage

```
reportNPAV(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

model	the model of the np.anova
dv	the name of the dependent variable that should be reported
write_to_clipboard	whether to write to the clipboard
sink_to	optional path of a .tex file to write the sentences to, so a manuscript can \input{} them

Details

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: \newcommand{\F}[3]{$F({#1},{#2})={#3}$} \newcommand{\p}{\textit{p=}} \newcommand{\pminor}{\text

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via message().

Examples

```
model <- data.frame(
  Df = c(1, 1, 10),
  `F value` = c(6.12, 5.01, NA),
  `Pr(>F)` = c(0.033, 0.045, NA),
  check.names = FALSE
)
rownames(model) <- c("Video", "gesture:eHMI", "Residuals")
reportNPAV(model, dv = "mental workload")
```

---

report_all *Analyze and report several dependent variables at once*

Description

Runs analyze_and_report() for each dependent variable (e.g., all questionnaire scales of a study) and additionally returns a summary table of the omnibus tests with Holm-adjusted p-values across the dependent variables, plus – when **patchwork** is installed – a combined figure.

Usage

```
report_all(
  data,
  dvs,
  iv,
  design = c("between", "within"),
  labels = NULL,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_dir = NULL
)
```

Arguments

data	the data frame
dvs	character vector of dependent variable column names
iv	the independent variable (column name as string)
design	"between" (default) or "within"
labels	optional named character vector mapping a dv name to its axis label, e.g. c(tlx_mental = "Mental Demand")
xlabels	optional labels for the x-axis, passed to every plot
plotType	either "box", "violin", or "boxviolin" (default)
sink_dir	optional directory; each dv's sentences are written to <sink_dir>/<dv>.tex so a manuscript can \input{} them

Value

Invisibly returns a list with components results (named list of analyze_and_report() results), summary (data frame with one row per dv: method, statistic, p.value, and Holm-adjusted p.holm), and combined_plot (a patchwork figure, or NULL when patchwork is not installed).

Examples

```
out <- report_all(mtcars, dvs = c("mpg", "disp"), iv = "cyl")
out$summary
```

---

reshape_data *Reshape Excel Data Based on Custom Markers and Include Custom ID Column*

Description

This function takes an Excel file with data in a wide format and transforms it to a long format. It includes a customizable "ID" column in the first position and repeats it for each slice. The function identifies sections of columns between markers that start with a user-defined string (default is "videoinfo") and appends those sections under the first section, aligning by column index.

Usage

```
reshape_data(
  input_filepath,
  sheetName = "Results",
  marker = "videoinfo",
  id_col = "ID",
  output_filepath
)
```

Arguments

input_filepath	String, the file path of the input Excel file.
sheetName	String, the name of the sheet to read from the Excel file. Default is "Results".
marker	String, the string that identifies the start of a new section of columns. Default is "videoinfo".
id_col	String, the name of the column to use as the ID column. Default is "ID".
output_filepath	String, the file path for the output Excel file.

Details

Relevant if you receive data in wide-format but cannot use built-in functionality due to naming (e.g., in LimeSurvey)

Value

None, writes the reshaped data to an Excel file specified by output_filepath.

Examples

```
if (requireNamespace("writexl", quietly = TRUE) &&
  requireNamespace("readxl", quietly = TRUE)) {
  tmp_in  <- tempfile(fileext = ".xlsx")
  tmp_out <- tempfile(fileext = ".xlsx")

  # Two marker-delimited sections of equal width; each section is stacked
  # under the first one, keyed by the ID column.
  toy <- data.frame(
    ID = c(1, 2),
    videoinfo1 = c("marker", "marker"),
    rating = c(10, 11),
    videoinfo2 = c("marker", "marker"),
    rating2 = c(20, 21),
    stringsAsFactors = FALSE
  )

  writexl::write_xlsx(toy, tmp_in)

  reshape_data(
```

```
    input_filepath = tmp_in,
    marker = "videoinfo",
    id_col = "ID",
    output_filepath = tmp_out
  )

  out <- readxl::read_excel(tmp_out)
  print(out)
}
```

`reverse_code` *Reverse-code responses*

Description

Flips a response scale so that a negatively worded item points the same way as the rest of its subscale: `min + max - x`.

Usage

```
reverse_code(x, min, max)
```

Arguments

`x`	Numeric responses.
`min, max`	The endpoints of the response scale the item was answered on – the *possible* range, not the observed one. Taking them from the data is the classic reverse-coding bug: if nobody picked 1, the flip is off by a point for every respondent.

Value

`x`, reverse-coded, with `NA` preserved.

Examples

```
reverse_code(c(1, 3, 5, NA), min = 1, max = 5)
```

`rFromNPAV` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements.

Description

Necessary LaTeX command: `\newcommand{\effectsize}{\textit{r=}}`

Usage

```
rFromNPAV(pvalue, N)
```

Arguments

`pvalue`	p value
`N`	number of measurements in the experiment

Value

Invisibly returns a list with components:

- `r`: effect size as a numeric scalar.
- `z`: corresponding z-statistic.
- `text`: LaTeX-formatted character string that is also sent to the console.

Examples

```
rFromNPAV(0.02, N = 180)
```

`rFromWilcox` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements.

Description

Calculation based on Rosenthal's formula (1994). N stands for the *number of measurements*.

Usage

```
rFromWilcox(wilcoxModel, N)
```

Arguments

`wilcoxModel`	the Wilcox model
`N`	number of measurements in the experiment

Value

Invisibly returns a list with components:

- r: effect size as a numeric scalar.
- z: corresponding z-statistic.
- text: character string that is also sent to the console.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
rFromWilcox(w, N = nrow(d))
```

---

rFromWilcoxAdjusted *rFromWilcoxAdjusted*

Description

rFromWilcoxAdjusted

Usage

```
rFromWilcoxAdjusted(wilcoxModel, N, adjustFactor)
```

Arguments

wilcoxModel	the Wilcox model
N	number of measurements in the experiment
adjustFactor	ad adjustment factor

Value

Invisibly returns a list with components:

- r: adjusted effect size as a numeric scalar.
- z: adjusted z-statistic.
- text: character string that is also sent to the console.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
rFromWilcoxAdjusted(w, N = nrow(d), adjustFactor = 2)
```

`save_paper_figure` *Save a plot with publication-ready defaults*

Description

Saves a ggplot with sizes matching common two-column conference/journal layouts (e.g., ACM): a single-column figure is 3.33 in wide, a full-width figure 7 in. On Windows and Linux, PDFs are rendered with `grDevices::cairo_pdf` so that fonts are embedded and unicode glyphs survive; on macOS the default pdf device is used instead, because R's cairo on macOS is known to crash some setups (e.g., GitHub Actions runners) and the macOS device handles fonts well on its own.

Usage

```
save_paper_figure(
  plot = ggplot2::last_plot(),
  filename,
  columns = 1,
  width = NULL,
  height = NULL,
  base_size = NULL,
  dpi = 300,
  device = NULL
)
```

Arguments

`plot`	The plot to save (defaults to the last plot displayed).
`filename`	Output path; the extension selects the device (`.pdf` is recommended for LaTeX).
`columns`	1 for a single-column figure, 2 for a full-width figure. Ignored when `width` is given.
`width`	Figure width in inches; overrides `columns`.
`height`	Figure height in inches. Defaults to 2/3 of the width.
`base_size`	Base font size in points for the saved figure, applied via `colley_theme()` sizing on top of whatever theme the plot carries. The default `NULL` derives it from `width`, so a 3.33 in figure gets about 7 pt and a 7 in figure about 9 pt – close to the body text of a typical two-column paper, which is what makes a figure legible

	at 100\ rather than only when zoomed. Pass a number to choose it yourself, or NA to leave the plot's own text sizes untouched.
dpi	Resolution for raster output. Default 300.
device	Graphics device passed to ggplot2::ggsave(). The default NULL selects it automatically as described above; pass e.g. grDevices::cairo_pdf explicitly to override.

Value

Invisibly returns filename.

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl), mpg)) +
  ggplot2::geom_boxplot()
save_paper_figure(p, file.path(tempdir(), "cyl-mpg.pdf"), columns = 1)
```

---

score_questionnaire *Score a standardised questionnaire*

Description

Turns the raw item columns of a published questionnaire into its subscale scores, applying that instrument's own scoring key: reverse-coding, any recoding it prescribes (centring a semantic differential to $-3..+3$, zero-basing the SUS), the subscale structure, and the published weights or multipliers. Supported instruments are listed by list_questionnaires(); register your own with define_questionnaire().

Usage

```
score_questionnaire(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
  reverse_items = NULL,
  min_valid = 1,
  append = FALSE,
  prefix_out = NULL,
  verbose = TRUE
)
```

Arguments

`data`	A data frame with one row per respondent (or per respondent and condition) and one column per item.
`instrument`	Instrument key, e.g. `"sus"`, `"nasa_tlx"`, `"ueq_s"`, `"tia"`, `"ssq"`. See `list_questionnaires()`.
`items`	Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (`c(mental = "tlx_md", effort = "tlx_ef", ...)`), which is order-proof and the safer choice for a survey export you did not lay out yourself.
`prefix`	Optional column-name prefix selecting the item columns, e.g. `"sus_"`. Matching columns are sorted numerically, so `sus_2` comes before `sus_10`. Must select exactly as many columns as the instrument has items.
`scale`	Optional two-element vector giving the response range your survey used, e.g. `c(1, 21)` for the 21-point NASA-TLX sheet or `c(0, 4)` for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale.
`reverse_items`	Optional items to *toggle* the reverse-coding of, given as item numbers or item codes. Naming an item the scoring key already reverses un-reverses it, which is what an export that stores a pair the other way round needs.
`min_valid`	Minimum proportion of a subscale's items that must be answered for a score to be produced. The default `1` scores only complete subscales and returns `NA` otherwise – no silent imputation. Relax it (e.g. `0.8`) to score partially complete responses, in which case a subscale is the mean of the items present, and a sum-scored instrument is scaled up proportionally so it stays on its published range.
`append`	Logical. If `TRUE`, return `data` with the score columns added; if `FALSE` (default), return only the scores.
`prefix_out`	Optional string prefixed to every score column, useful when the same instrument is scored more than once per row (pre/post, or one block per condition).
`verbose`	Logical. If `TRUE` (default), emit the one-time mapping message described above.

Value

A data frame with one row per row of `data` and one column per subscale, plus the instrument's overall score where it defines one. The instrument definition and the resolved column mapping are attached as the `"instrument"` and `"mapping"` attributes.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run check_questionnaire() once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the "mapping" attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with options(colleyRstats.quiet_questionnaires = TRUE)).

A named items argument (items = c(mental = "tlx_md", ...)) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself.

See Also

check_questionnaire() to verify the mapping, score_reliability() for internal consistency, questionnaire_items() for the item list, define_questionnaire() to add an instrument.

Examples

```
set.seed(42)
sus_raw <- as.data.frame(matrix(sample(1:5, 10 * 6, TRUE), nrow = 6))
names(sus_raw) <- paste0("sus_", 1:10)

score_questionnaire(sus_raw, "sus", prefix = "sus_")

# A 21-point NASA-TLX sheet, scored onto the conventional 0-100
tlx <- data.frame(
  mental = c(14, 3), physical = c(2, 1), temporal = c(11, 4),
  performance = c(6, 2), effort = c(13, 5), frustration = c(9, 1)
)
score_questionnaire(tlx, "nasa_tlx", scale = c(1, 21))
```

---

score_reliability *Internal consistency of a questionnaire's subscales*

Description

Cronbach's alpha per subscale, computed on the same recoded item matrix that score_questionnaire() aggregates – so reverse-coded items are already flipped, and a negative alpha means a genuine problem rather than a forgotten reversal. McDonald's omega is added when **psych** is installed, and is the better-behaved statistic when a subscale's items are not equally good indicators.

Usage

```
score_reliability(
  data,
  instrument,
  items = NULL,
  prefix = NULL,
  scale = NULL,
  reverse_items = NULL,
  verbose = TRUE
)
```

Arguments

data	A data frame with one row per respondent (or per respondent and condition) and one column per item.
instrument	Instrument key, e.g. "sus", "nasa_tlx", "ueq_s", "tia", "ssq". See list_questionnaires().
items	Optional. Either the item columns in the instrument's own order, or a character vector *named* after the item codes (c(mental = "tlx_md", effort = "tlx_ef", ...)), which is order-proof and the safer choice for a survey export you did not lay out yourself.
prefix	Optional column-name prefix selecting the item columns, e.g. "sus_". Matching columns are sorted numerically, so sus_2 comes before sus_10. Must select exactly as many columns as the instrument has items.
scale	Optional two-element vector giving the response range your survey used, e.g. c(1, 21) for the 21-point NASA-TLX sheet or c(0, 4) for a zero-based SUS. Responses are rescaled onto the instrument's own range before scoring. Defaults to the instrument's range; responses outside it are an error rather than a silent rescale.
reverse_items	Optional items to *toggle* the reverse-coding of, given as item numbers or item codes. Naming an item the scoring key already reverses un-reverses it, which is what an export that stores a pair the other way round needs.
verbose	Logical. If TRUE (default), emit the one-time mapping message described above.

Value

A data frame with one row per subscale: the number of items, the number of complete cases it was computed on, alpha, omega (NA without **psych**), and the mean inter-item correlation. Subscales of a single item yield NA – internal consistency is not defined for them.

Caution – verify the mapping against your own survey

Item numbers, item order and item polarity are properties of the sheet a study actually administered, not of the instrument in the abstract. Survey tools renumber items, translations reorder them, short forms drop them from the middle, and semantic differentials get printed with the poles the other way round.

This package applies each instrument's **published** scoring key, which is the right default and is still only a default. If the sheet your participants saw differed, the scores will be wrong – and wrong quietly, because a mismatched mapping raises no error and produces entirely plausible numbers.

So: run check_questionnaire() once per instrument per study and read the mapping it prints, and double-check any figure before it goes into a paper. The mapping used is also attached to the result as the "mapping" attribute, and summarised in a console note the first time each distinct mapping is scored in a session (silence it with options(colleyRstats.quiet_questionnaires = TRUE)).

A named items argument (items = c(mental = "tlx_md", ...)) removes the positional assumption altogether and is the safer choice for an export you did not lay out yourself.

See Also

score_questionnaire()

Examples

```
set.seed(3)
trait <- rnorm(60)
d <- as.data.frame(lapply(1:10, function(i) {
  round(pmin(pmax(3 + trait + rnorm(60, sd = 0.6), 1), 5))
}))
names(d) <- paste0("sus_", 1:10)
# Items 2, 4, 6, 8, 10 are negatively worded on the real SUS, so flip them
d[paste0("sus_", c(2, 4, 6, 8, 10))] <- 6 - d[paste0("sus_", c(2, 4, 6, 8, 10))]
score_reliability(d, "sus", prefix = "sus_")
```

---

stat_sum_df *Generating the sum and adding a crossbar.*

Description

Generating the sum and adding a crossbar.

Usage

```
stat_sum_df(fun, geom = "crossbar", ...)
```

Arguments

fun	function
geom	geom to be shown
...	Additional arguments passed to stat_summary

Value

A ggplot2 layer that can be added to a ggplot object.

Examples

```
# Simple summary function: use the mean as y, ymin, and ymax
mean_fun <- function(x) {
  m <- mean(x, na.rm = TRUE)
  data.frame(y = m, ymin = m, ymax = m)
}

ggplot2::ggplot(mtcars, ggplot2::aes(x = factor(cyl), y = mpg)) +
  stat_sum_df(mean_fun)
```

summarize_sickness *Summarise a motion-sickness time course*

Description

Repeated single-item sickness ratings (FMS, MISC) are collected once a minute and then, almost always, reduced to a handful of per-participant numbers before they are analysed. This produces those numbers: peak, mean, final rating, the area under the rating curve, and when a threshold was first crossed.

Usage

```
summarize_sickness(data, value, id, time = NULL, by = NULL, threshold = NULL)
```

Arguments

data	A data frame in long format: one row per rating.
value	Column holding the rating.
id	Column identifying the participant (and, with by, the condition).
time	Optional column holding the time of the rating, in whatever unit the study used. Needed for a meaningful area under the curve and for time-to-threshold; without it, the rating index is used and auc is reported in rating-steps.
by	Optional further grouping columns, e.g. the experimental condition.
threshold	Optional rating at or above which a participant counts as affected, e.g. 6 for MISC (nausea) or 10 for FMS.

Value

A data frame with one row per participant (and by group): n ratings, peak, mean, final, auc (trapezoidal), auc_rate (the AUC divided by the observed duration, i.e. the time-weighted mean rating), and – with threshold – reached and time_to_threshold.

See Also

score_questionnaire() for the single-measurement instruments

Examples

```
d <- data.frame(
  pid = rep(c("p1", "p2"), each = 5),
  minute = rep(0:4, 2),
  fms = c(0, 1, 3, 6, 8, 0, 0, 1, 1, 2)
)
summarize_sickness(d, value = "fms", id = "pid", time = "minute", threshold = 5)
```

use_colleyrstats_sty *Write colleyRstats.sty into a project (for Overleaf)*

Description

Copies the **colleyRstats** LaTeX macro package next to your manuscript so the report output compiles with a single `\usepackage{colleyRstats}` – no need to paste `latex_preamble()` into the preamble. Upload the written `colleyRstats.sty` to your Overleaf project (or keep it in the same folder as `main.tex`).

Usage

```
use_colleyrstats_sty(dir = ".", overwrite = FALSE)
```

Arguments

`dir`	Directory to write `colleyRstats.sty` into. Default the current working directory.
`overwrite`	Overwrite an existing file? Default `FALSE`.

Value

Invisibly, the path to the written `.sty` file.

Examples

```
use_colleyrstats_sty(tempdir(), overwrite = TRUE)
```

use_study_project *Scaffold a reproducible study analysis*

Description

Writes a complete, runnable analysis project: a **targets** pipeline that recomputes only what changed, R scripts split along the stages every user study goes through (read, clean, score, model, plot), a Quarto report, and a directory the generated LaTeX lands in so a manuscript can `\input{}` the numbers instead of having them re-typed.

Usage

```
use_study_project(
  path,
  name = NULL,
  questionnaires = c("nasa_tlx", "sus"),
  renv = requireNamespace("renv", quietly = TRUE),
  git = TRUE,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`path`	Directory to create the project in. Created if it does not exist.
`name`	Project name, used in the README and the report title. Defaults to the directory name.
`questionnaires`	Character vector of instrument keys the study uses, e.g. `c("nasa_tlx", "sus")`. The scoring script is generated with one call per instrument. See `list_questionnaires()`.
`renv`	Logical. Initialise **renv** in the project, pinning the package versions this analysis was run with. Default `TRUE` when **renv** is installed. This is what makes the project still run in three years, and what makes it a usable open-science artifact.
`git`	Logical. Write a `.gitignore` suited to an R analysis project. Default `TRUE`.
`overwrite`	Logical. Replace files that already exist. Default `FALSE`, so running this on a live project adds missing pieces without touching your work.
`quiet`	Logical. Suppress the per-file messages.

Details

The generated pipeline is wired to this package: scoring goes through `score_questionnaire()`, model choice and fitting through `fit_recommended()`, sentences and tables through the `report_*` functions, and figures through `save_paper_figure()`. It runs as written against the example data it ships with, so the first thing you do in a new project is see a green pipeline, then replace the example data with yours.

Value

Invisibly, the normalised project path.

Why a pipeline rather than a script

A study analysis is re-run many times – after a data fix, after a reviewer asks for one more contrast, after a co-author changes a factor label. With a script, that means re-running everything and hoping nothing stale is left in the workspace; **targets** tracks which step depends on what and recomputes only the affected ones, which also means the pipeline is a machine-checkable record of how each number was produced.

See Also

score_questionnaire(), fit_recommended(), save_paper_figure(), emit_overleaf()

Examples

```
project <- file.path(tempdir(), "driving-study")
use_study_project(project, questionnaires = c("nasa_tlx", "sus"), renv = FALSE)
list.files(project, recursive = TRUE)
```

Index

%!in% *(not_in)*, 44

add_pareto_emoa_column, 3
add_pareto_emoa_column(), *5*
add_pareto_moocore_column, 4
add_pareto_moocore_column(), *4*
analyze_and_report, 6
analyze_and_report(), *23*, *26*, *68*, *69*
animate_mobo2, 7
ARTool::art.con(), *52*, *54*
assumption_methods_text, 9
assumption_methods_text(), *6*, *11*

check_assumptions_anova *(checkAssumptionsForAnova)*, 9
check_homogeneity_by_group, 10
check_normality_by_group, 11
check_questionnaire, 12
check_questionnaire(), *13*, *19*, *20*, *46*, *77*, *78*
checkAssumptionsForAnova, 9
cite_methods, 13
cite_methods(), *23*
classify_outcome, 14
classify_outcome(), *26*, *47*, *48*
colley_theme, 17
colley_theme(), *15*, *74*
colleyRstats_setup, 15

debug_contr_error, 18
define_questionnaire, 19
define_questionnaire(), *42*, *75*, *77*
define_result_macro, 21

emit_name_macros, 22
emit_overleaf, 22
emit_overleaf(), *83*
expand_latex_macros, 23

figure_base_size, 24
fit_recommended, 25
fit_recommended(), *82*, *83*

generateEffectPlot, 27
generateMoboPlot, 29
generateMoboPlot2, 31
generateMoboPlot2(), *7*, *8*
ggbetweenstatsWithPriorNormalityCheck, 33
ggbetweenstatsWithPriorNormalityCheckAsterisk, 35
ggplot2::ggsave(), *75*
ggwithinstatsWithPriorNormalityCheck, 36
ggwithinstatsWithPriorNormalityCheckAsterisk, 38

latex_escape, 41
latex_preamble, 41
latex_preamble(), *24*, *64*, *81*
latexify_report, 39
list_questionnaires, 42
list_questionnaires(), *12*, *20*, *46*, *75*, *76*, *78*, *82*

n_fun, 45
na.zero, 42
normalize, 43
not_empty, 44
not_in, 44
nparLD, *66*

pathPrep, 45
plot_between_stats *(ggbetweenstatsWithPriorNormalityCheck)*, 33
plot_between_stats_asterisk *(ggbetweenstatsWithPriorNormalityCheckAsterisk)*, 35
plot_effect *(generateEffectPlot)*, 27
plot_mobo *(generateMoboPlot)*, 29

plot_mobo2 (generateMoboPlot2), 31
plot_within_stats
(ggwithinstatsWithPriorNormalityCheck), 36
plot_within_stats_asterisk
(ggwithinstatsWithPriorNormalityCheck), 38

questionnaire_items, 46
questionnaire_items(), *13*, *42*, *77*

recommend_analysis (recommend_test), 47
recommend_test, 47
recommend_test(), *14*, *25*, *26*
remove_outliers_REI, 48
replace_values, 49
report_all, 68
report_all(), *23*
report_art (reportART), 50
report_art_con (reportArtCon), 52
report_art_con_table
(reportArtConTable), 53
report_clmm (reportCLMM), 56
report_dunn_test (reportDunnTest), 57
report_dunn_test_table
(reportDunnTestTable), 58
report_ggstatsplot (reportggstatsplot), 60
report_ggstatsplot_posthoc
(reportggstatsplotPostHoc), 62
report_glmm (reportGLMM), 63
report_mean_sd (reportMeanAndSD), 65
report_nparld (reportNparLD), 66
reportART, 50
reportArtCon, 52
reportArtConTable, 53
reportCLMM, 56
reportDunnTest, 57
reportDunnTest(), *52*
reportDunnTestTable, 58
reportDunnTestTable(), *53*
reportggstatsplot, 60
reportggstatsplot(), *6*
reportggstatsplotPostHoc, 62
reportggstatsplotPostHoc(), *6*
reportGLMM, 63
reportGLMM(), *56*
reportMeanAndSD, 65
reportNparLD, 66
reportNPAV, 67
reshape_data, 69
reverse_code, 71
rFromNPAV, 72
rFromWilcox, 72
rFromWilcoxAdjusted, 73

save_paper_figure, 74
save_paper_figure(), *15*, *17*, *23*, *24*, *82*, *83*
score_questionnaire, 75
score_questionnaire(), *12*, *13*, *19*, *20*, *42*, *46*, *77*, *78*, *80*, *82*, *83*
score_reliability, 77
score_reliability(), *19*, *77*
stat_sum_df, 79
summarize_sickness, 80

use_colleyrstats_sty, 81
use_colleyrstats_sty(), *41*
use_study_project, 81