

Package ‘specr’

October 16, 2026

Title Conducting and Visualizing Specification Curve Analyses

Version 1.1.0

Description Provides utilities for conducting specification curve analyses (Simonsohn, Simmons & Nelson, 2020, <[doi:10.1038/s41562-020-0912-z](https://doi.org/10.1038/s41562-020-0912-z)>) or multiverse analyses (Steen-
gen, Tuerlinckx, Gelman & Vanpaemel, 2016, <[doi:10.1177/1745691616658637](https://doi.org/10.1177/1745691616658637)>) includ-
ing functions to setup, run, evaluate, and plot all specifications, as well as to conduct infer-
ence on the specification curve based on resampling under the null.

License GPL-3

URL <https://masurp.github.io/specr/>, <https://github.com/masurp/specr>

BugReports <https://github.com/masurp/specr/issues>

Depends R (>= 4.1.0)

Imports broom, cowplot, dplyr (>= 1.1.0), furrr, future, ggplot2 (>= 3.4.0), ggraph, igraph, lifecycle (>= 1.0.2), lme4, magrittr, methods, purrr, rlang, stringr, tibble, tidyr (>= 1.0.0)

Suggests broom.mixed, fixest, gapminder, knitr, lavaan, testthat (>= 3.0.0), tidyverse, performance, rmarkdown, survival, withr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Philipp K. Masur [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-3065-7305>>),
Michael Scharkow [aut]

Maintainer Philipp K. Masur <phil.masur@gmail.com>

Repository CRAN

Date/Publication 2026-10-16 08:50:08 UTC

Contents

example_data	2
icc_specs	3
infer_curve	4
plot.spectr.infer	6
plot.spectr.object	7
plot.spectr.setup	10
plot_choices	11
plot_curve	12
plot_decisiontree	14
plot_samplesizes	15
plot_specs	16
plot_summary	17
plot_variance	18
run_specs	20
setup	21
spectr	25
summarise_specs	28
summary.spectr.infer	29
summary.spectr.object	31
summary.spectr.setup	33
Index	35

example_data	<i>Example data set</i>
--------------	-------------------------

Description

This simulated data set can be used to explore the major function of 'spectr'. It provides variables that can be used to mimic different independent and dependent variables, control variables, and grouping variables (for subset analyses).

Usage

```
data(example_data)
```

Format

- A tibble with 1,000 rows and 15 variables (no missing values):
- group1** Grouping variable with three groups ("young", "middle", "old"), e.g. for subset analyses.
 - group2** Grouping variable with two groups ("female", "male").
 - group3** Grouping variable with three groups ("A", "B", "C").
 - x1, x2, x3, x4** Numeric variables that can be used as (alternative) independent variables.
 - c1, c2, c3, c4** Numeric variables that can be used as control variables.
 - y1, y2, y3, y4** Numeric variables that can be used as (alternative) dependent variables.

Source

Simulated data.

Examples

```
data(example_data)
head(example_data)
```

icc_specs	<i>Compute intraclass correlation coefficient</i>
-----------	---

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can decompose the variance simply by using the `generic_plot()` function and adding the argument `type = "variance"`, which also prints the variance components. This function extracts intraclass correlation coefficients (ICC) from a multilevel model. It can be used to decompose the variance in the outcome variable of a specification curve analysis (e.g., the regression coefficients). This approach summarises the relative importance of analytical choices by estimating the share of variance in the outcome (e.g., the regression coefficient) that different analytical choices or combinations thereof account for. To use this approach, one needs to estimate a multilevel model that includes all analytical choices as grouping variables (see examples).

Usage

```
icc_specs(model, percent = TRUE)
```

Arguments

<code>model</code>	a multilevel (i.e., mixed effects) model that captures the variances of the specification curve.
<code>percent</code>	a logical value indicating whether the ICC should also be printed as percentage. Defaults to <code>TRUE</code> .

Value

a data frame including the grouping variable, the random effect variances, the raw intraclass correlation coefficient (ICC), and the ICC in percent.

References

- Hox, J. J. (2010). Multilevel analysis: techniques and applications. New York: Routledge.

See Also

[plot_variance\(\)](#) to plot the variance decomposition.

Examples

```
# Step 1: Run spec curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"))

# Step 2: Estimate a multilevel model without predictors
model <- lme4::lmer(estimate ~ 1 + (1|x) + (1|y), data = results)

# Step 3: Estimate intra-class correlation
icc_specs(model)
```

infer_curve

Inference on the specification curve

Description

Tests how inconsistent the specification curve as a whole is with the null hypothesis that the independent variable has no effect in any of the specifications. Following Simonsohn, Simmons and Nelson (2020, section "Inference with non-experimental data"), the function generates the distribution of specification curves that would be expected under the null by resampling data in which the null hypothesis is known to be true. The resulting object of class `spectr.infer` can be summarized with `summary()` (test statistics and p-values) and plotted with `plot()` (observed curve against the curves under the null).

Usage

```
infer_curve(x, data, n_samples = 500, fast = TRUE, ...)
```

Arguments

<code>x</code>	A <code>spectr</code> object, resulting from running <code>spectr()</code> (or the tibble that <code>spectr()</code> returns when it is given a tibble of specifications).
<code>data</code>	The data set that was used to fit the specifications, or the <code>spectr.setup</code> object created by <code>setup()</code> (whose data set is then used).
<code>n_samples</code>	Number of bootstrap samples drawn under the null. Defaults to 500. Use at least 1,000 for results that are reported.
<code>fast</code>	Logical value indicating whether specifications that were fitted with <code>lm</code> should be refitted with a fast least-squares routine. It gives the same estimates and p-values as <code>lm()</code> , which is checked against the observed results for every specification before it is used. Set to <code>FALSE</code> to refit all specifications with their original model function. Defaults to <code>TRUE</code> .
<code>...</code>	Further arguments passed to <code>furrr::future_map()</code> , e.g. <code>.progress = TRUE</code> . The argument <code>.options</code> cannot be set, because the resampling always uses <code>furrr_options(seed = TRUE)</code> .

Details

Procedure

For every specification k , the dependent variable is first modified so that the null hypothesis is true by construction: $y_k^* = y_k - b_k x_k$, where b_k is the estimate observed in that specification. Then, in each of `n_samples` bootstrap samples, N rows are drawn with replacement from the data (N being the number of rows of data), and every specification is refitted on these rows with y_k^* as its dependent variable. The same rows are used for all specifications, which preserves the dependence between them. Each specification keeps its own model, control variables, subset, and additions to the formula, so that the curves under the null are based on exactly the same specifications as the observed curve.

Requirements

Because the null is imposed by subtracting $b_k x_k$ from the dependent variable, the procedure is only valid for models that are linear in the dependent variable (e.g., `lm`, `lmer`, or `glm` with a gaussian family and identity link), for numeric independent and dependent variables, and for unstandardized estimates. Before resampling, `infer_curve()` refits every specification on the full data with y_k^* and stops if the estimate is not zero, which catches models and extraction functions for which this does not hold (e.g., logistic regression). Structural equation models (`lavaan`) are not supported. Note that rows are resampled independently, which ignores clustering (e.g., in multilevel data).

Interpretation

Joint inference across all specifications is only meaningful if the specifications are genuinely equivalent, arbitrary alternatives (Del Giudice & Gangestad, 2021). See `vignette("inferences")` for a discussion.

Parallelization

The bootstrap samples are distributed across workers if a parallel plan has been set up front, e.g. `future::plan(future::multisession, workers = 4)`. Results are reproducible with `set.seed()` and identical whether they are computed sequentially or in parallel. For more information, see `vignette("parallelization")`.

Value

An object of class `spectr.infer`, a list that includes the observed curve (`observed`), the estimates and p-values under the null (`estimates` and `p_values`; matrices with one row per bootstrap sample and one column per specification), and technical information. Use `methods(class = "spectr.infer")` for an overview on available methods.

References

- Simonsohn, U., Simmons, J.P. & Nelson, L.D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. <https://doi.org/10.1038/s41562-020-0912-z>
- Del Giudice, M., & Gangestad, S. W. (2021). A traveler's guide to the multiverse: Promises, pitfalls, and a framework for the evaluation of analytic decisions. *Advances in Methods and Practices in Psychological Science*, 4(1). <https://doi.org/10.1177/2515245920954925>

See Also

`summary.spectr.infer()` and `plot.spectr.infer()` to investigate the results.

Examples

```
# Setup and run specification curve analysis
specs <- setup(data = example_data,
  y = "y1",
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group2 = c("female", "male")))

results <- spectr(specs)

# Resample under the null (better 1,000 or more)
set.seed(42)
inference <- infer_curve(results, specs, n_samples = 200)

# Test statistics and p-values
summary(inference)

# Observed curve against the curves under the null
plot(inference)

# For comparison: the same analysis after shuffling y1, which removes any
# effect of x1 and x2 in all specifications
shuffled <- example_data
shuffled$y1 <- sample(shuffled$y1)
specs_null <- setup(data = shuffled,
  y = "y1",
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group2 = c("female", "male")))

summary(infer_curve(spectr(specs_null), specs_null, n_samples = 200))
```

plot.spectr.infer

Plot specification curve against curves under the null

Description

This function plots the observed specification curve (estimates ranked from lowest to highest) on top of the median and the 2.5% and 97.5% quantiles of the ranked curves under the null (resulting from `infer_curve()`).

Usage

```
## S3 method for class 'spectr.infer'
plot(x, ...)
```

Arguments

`x` A `spectr.infer` object, resulting from calling `infer_curve()`.
`...` further arguments passed to or from other methods (currently ignored).

Value

A `ggplot` object that can be customized further.

See Also

`infer_curve()`, `summary.spectr.infer()`

Examples

```
specs <- setup(data = example_data,
  y = "y1",
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"))

set.seed(42)
inference <- infer_curve(spectr(specs), specs, n_samples = 200) # better 1,000 or more

# Observed curve against the curves under the null
plot(inference)
```

plot.spectr.object	<i>Plot specification curve and analytic choices</i>
--------------------	--

Description

This function plots visualizations of the specification curve analysis. The function requires an object of class `spectr.object`, usually the results of calling `spectr()` to create a standard visualization of the specification curve analysis. Several types of visualizations are possible.

Usage

```
## S3 method for class 'spectr.object'
plot(
  x,
  type = "default",
  var = .data$estimate,
  group = NULL,
  choices = c("x", "y", "model", "controls", "subsets"),
  labels = c("A", "B"),
  rel_heights = c(2, 3),
  desc = FALSE,
  null = 0,
```

```

ci = TRUE,
ribbon = FALSE,
formula = NULL,
print = TRUE,
choice_labels = c("right", "left"),
...
)

```

Arguments

<code>x</code>	A <code>spcr.object</code> object, usually resulting from calling <code>spcr()</code> .
<code>type</code>	What type of figure should be plotted? If <code>type = "default"</code> , the standard specification curve analysis plot (the specification curve as the upper panel and an overview of the relevant choices as the lower panel) is created. If <code>type = "curve"</code> , only the specification curve (upper panel of the default plot) is plotted. If <code>type = "choices"</code> , only the choice panel (lower part of the default plot) is plotted. If <code>type = "boxplot"</code> , an alternative visualization of differences between choices is plotted that summarizes results per choice using box-and-whisker plot(s). If <code>type = "variance"</code> , the variance of the estimates is decomposed into the shares explained by the analytical choices (see <code>formula</code>). If <code>type = "samplesizes"</code> , a barplot of sample sizes per specification is plotted. See examples for more information.
<code>var</code>	Which parameter should be plotted in the curve? Defaults to <code>estimate</code> , but other parameters (e.g., <code>p.value</code> , <code>fit_r.squared</code> ,...) can be plotted too. Note that the confidence intervals (<code>ci</code>) and the colors always refer to the estimate, so set <code>ci = FALSE</code> for other parameters.
<code>group</code>	Should the arrangement of the curve be grouped by a particular choice? Defaults to <code>NULL</code> , but can be any of the present choices (e.g., <code>x</code> , <code>y</code> , <code>controls</code> ...), given unquoted (<code>group = x</code>) or as a string (<code>group = "x"</code>).
<code>choices</code>	A vector specifying which analytic choices should be plotted. By default, all choices (<code>x</code> , <code>y</code> , <code>model</code> , <code>controls</code> , <code>subsets</code>) are plotted.
<code>labels</code>	Labels for the two parts of the plot
<code>rel_heights</code>	vector indicating the relative heights of the plot.
<code>desc</code>	Logical value indicating whether the curve should be arranged in a descending order. Defaults to <code>FALSE</code> .
<code>null</code>	Indicate what value represents the 'null' hypothesis (defaults to zero). Specifications whose confidence intervals exclude this value are highlighted (positive = blue, negative = red).
<code>ci</code>	Logical value indicating whether confidence intervals should be plotted.
<code>ribbon</code>	Logical value indicating whether a ribbon instead should be plotted
<code>formula</code>	In combination with <code>type = "variance"</code> , you can provide a specific formula to extract specific variance components. The syntax of the formula is based on <code>lme4::lmer()</code> and thus looks something like, e.g.: <code>"estimate ~ 1 + (1 x) + (1 y)"</code> (to estimate the amount of variance explained by different independent <code>x</code> and dependent variables <code>y</code>). All other choices are then subsumed under

	residual variance. If no formula is provided, all choices (x, y, model, controls, and subsets) that have more than one alternative are included. See examples for further details.
print	In combination with type = "variance", logical value indicating whether the intra-class correlations (i.e., percentages of variance explained by analytical choices) should be printed or not. Defaults to TRUE.
choice_labels	Where should the names of the analytical choices (x, y, model, ...) be shown in the panel of choices (type = "default" or "choices")? Either "right" (default) or "left", next to their options.
...	further arguments passed to or from other methods (currently ignored).

Value

A [ggplot](#) object that can be customized further.

Examples

```
# Specification Curve analysis ----
# Setup specifications
specs <- setup(data = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# Run analysis
results <- spectr(specs)

# Resulting data frame with estimates
as_tibble(results) # This will be used for plotting

# Visualizations ---
# Plot results in various ways
plot(results) # default
plot(results, choices = c("x", "y")) # specific choices
plot(results, ci = FALSE, ribbon = TRUE) # exclude CI and add ribbon instead
plot(results, type = "curve")
plot(results, type = "choices")
plot(results, type = "samplesizes")
plot(results, type = "boxplot")

# Names of the choices on the left, next to their options
plot(results, choice_labels = "left")

# Grouped plot
plot(results, group = controls)
```

```

# Alternative and specific visualizations ----
# Other variables in the resulting data set can be plotted too
plot(results,
      type = "curve",
      var = fit_r.squared, # extract "r-square" instead of "estimate"
      ci = FALSE)

# Such a plot can also be extended (e.g., by again adding the estimates with
# confidence intervals)
library(ggplot2)
plot(results, type = "curve", var = fit_r.squared) +
  geom_point(aes(y = estimate), shape = 5) +
  labs(x = "specifications", y = "r-squared | estimate")

# We can also investigate how much variance is explained by each analytical choice
plot(results, type = "variance")

# By providing a specific formula in `lme4::lmer()`-style, we can extract specific choices
# and also include interactions between choices
plot(results,
      type = "variance",
      formula = "estimate ~ 1 + (1|x) + (1|y) + (1|group1) + (1|x:y)")

## Combining several plots ----
# `speccr` also exports the function `plot_grid()` from the package `cowplot`, which
# can be used to combine plots meaningfully
a <- plot(results, "curve")
b <- plot(results, "choices", choices = c("x", "y", "controls"))
c <- plot(results, "samplesizes")
plot_grid(a, b, c,
          align = "v",
          axis = "rbl",
          rel_heights = c(2, 3, 1),
          ncol = 1)

```

plot.speccr.setup

Plot visualization of the specification setup

Description

This function plots a visual summary of the specification setup. It requires an object of class `speccr.setup`, usually the result of calling `setup()`.

Usage

```

## S3 method for class 'speccr.setup'
plot(x, layout = "dendrogram", circular = FALSE, ...)

```

Arguments

x	A <code>specr.setup</code> object, usually resulting from calling <code>setup()</code> .
layout	The type of layout to create for the garden of forking path. Defaults to "dendrogram". See <code>?ggraph</code> for options.
circular	Should the layout be transformed into a radial representation. Only possible for some layouts. Defaults to FALSE.
...	further arguments passed to or from other methods (currently ignored).

Value

A [ggplot](#) object that can be customized further.

Examples

```
specs <- setup(data = example_data,
  x = c("x1", "x2", "x3"),
  y = c("y1", "y2"),
  model = c("lm", "glm"),
  controls = "c1",
  subsets = list(group2 = unique(example_data$group2)))

plot(specs)
plot(specs, circular = TRUE)
```

plot_choices

Plot how analytical choices affect results

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "choices"`. This functions plots how analytic choices affect the obtained results (i.e., the rank within the curve). Significant results are highlighted (negative = red, positive = blue, grey = nonsignificant). This functions creates the lower panel in `plot_specs()`.

Usage

```
plot_choices(
  df,
  var = .data$estimate,
  group = NULL,
  choices = c("x", "y", "model", "controls", "subsets"),
  desc = FALSE,
  null = 0
)
```

Arguments

df	a data frame resulting from run_specs().
var	which variable should be evaluated? Defaults to estimate (the effect sizes computed by run_specs()).
group	Should the arrangement of the curve be grouped by a particular choice? Defaults to NULL, but can be any of the present choices (e.g., x, y, controls...)
choices	a vector specifying which analytical choices should be plotted. By default, all choices are plotted.
desc	logical value indicating whether the curve should be arranged in a descending order. Defaults to FALSE.
null	Indicate what value represents the 'null' hypothesis (Defaults to zero).

Value

a `ggplot` object.

Examples

```
# Run specification curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"),
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# Plot simple table of choices
plot_choices(results)

# Plot only specific choices
plot_choices(results,
  choices = c("x", "y", "controls"))
```

plot_curve

Plot ranked specification curve

Description

[Deprecated] This function is deprecated because the new version of `spectr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "curve"`. This function plots the a ranked specification curve. Confidence intervals can be included. Significant results are highlighted (negative = red, positive = blue, grey = nonsignificant). This functions creates the upper panel in `plot_specs()`.

Usage

```
plot_curve(
  df,
  var = .data$estimate,
  group = NULL,
  desc = FALSE,
  ci = TRUE,
  ribbon = FALSE,
  legend = FALSE,
  null = 0
)
```

Arguments

df	a data frame resulting from <code>run_specs()</code> .
var	which variable should be evaluated? Defaults to estimate (the effect sizes computed by <code>run_specs()</code>).
group	Should the arrangement of the curve be grouped by a particular choice? Defaults to NULL, but can be any of the present choices (e.g., x, y, controls...)
desc	logical value indicating whether the curve should be arranged in a descending order. Defaults to FALSE.
ci	logical value indicating whether confidence intervals should be plotted.
ribbon	logical value indicating whether a ribbon instead should be plotted.
legend	logical value indicating whether the legend should be plotted. Defaults to FALSE. Currently has no effect, because the colors are set directly and no legend is created.
null	Indicate what value represents the null hypothesis (Defaults to zero)

Value

a `ggplot` object.

Examples

```
# load additional library
library(ggplot2) # for further customization of the plots

# Run specification curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"),
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# Plot simple specification curve
plot_curve(results)
```

```
# Ribbon instead of CIs and customize further
plot_curve(results, ci = FALSE, ribbon = TRUE) +
  geom_hline(yintercept = 0) +
  geom_hline(yintercept = median(results$estimate),
             linetype = "dashed") +
  theme_linedraw()
```

plot_decisiontree	<i>Plot decision tree</i>
-------------------	---------------------------

Description

[Deprecated] This function is deprecated because the new version of `spectr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()`. This function plots a simple decision tree that is meant to help understanding how few analytical choices may results in a large number of specifications. It is somewhat useless if the final number of specifications is very high.

Usage

```
plot_decisiontree(df, label = FALSE, legend = FALSE)
```

Arguments

<code>df</code>	data frame resulting from <code>run_specs()</code> .
<code>label</code>	Logical. Should labels be included? Defaults to FALSE. Produces only a reasonable plot if number of specifications is low.
<code>legend</code>	Logical. Should specific decisions be identifiable. Defaults to FALSE.

Value

a `ggplot` object.

Examples

```
results <- run_specs(df = example_data,
                    y = c("y1", "y2"),
                    x = c("x1", "x2"),
                    model = c("lm"),
                    controls = c("c1", "c2"))

# Basic, non-labelled decisions tree
plot_decisiontree(results)

# Labelled decisions tree
plot_decisiontree(results, label = TRUE)

# Add legend
plot_decisiontree(results, label = TRUE, legend = TRUE)
```

plot_samplesizes	<i>Plot sample sizes</i>
------------------	--------------------------

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "samplesizes"`. This function plots a histogram of sample sizes per specification. It can be added to the overall specification curve plot (see vignettes).

Usage

```
plot_samplesizes(df, var = .data$estimate, group = NULL, desc = FALSE)
```

Arguments

<code>df</code>	a data frame resulting from <code>run_specs()</code> .
<code>var</code>	which variable should be evaluated? Defaults to <code>estimate</code> (the effect sizes computed by <code>run_specs()</code>).
<code>group</code>	Should the arrangement of the curve be grouped by a particular choice? Defaults to <code>NULL</code> , but can be any of the present choices (e.g., <code>x</code> , <code>y</code> , <code>controls</code> ...)
<code>desc</code>	logical value indicating whether the curve should be arranged in a descending order. Defaults to <code>FALSE</code> .

Value

a `ggplot` object.

Examples

```
# load additional library
library(ggplot2) # for further customization of the plots

# run specification curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"),
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# plot ranked bar chart of sample sizes
plot_samplesizes(results)

# add a horizontal line for the median sample size
plot_samplesizes(results) +
  geom_hline(yintercept = median(results$fit_nobs),
    color = "darkgrey",
```

```
linetype = "dashed") +
theme_linedraw()
```

plot_specs

Plot specification curve and analytical choices

Description

[Deprecated] This function is deprecated because the new version of `spectr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "default"`. This function plots an entire visualization of the specification curve analysis. The function uses the entire [tibble](#) that is produced by `run_specs()` to create a standard visualization of the specification curve analysis. Alternatively, one can also pass two separately created [ggplot](#) objects to the function. In this case, it simply combines them using `cowplot::plot_grid`. Significant results are highlighted (negative = red, positive = blue, grey = nonsignificant).

Usage

```
plot_specs(
  df = NULL,
  plot_a = NULL,
  plot_b = NULL,
  choices = c("x", "y", "model", "controls", "subsets"),
  labels = c("A", "B"),
  rel_heights = c(2, 3),
  desc = FALSE,
  null = 0,
  ci = TRUE,
  ribbon = FALSE,
  ...
)
```

Arguments

<code>df</code>	a data frame resulting from <code>run_specs()</code> .
<code>plot_a</code>	a <code>ggplot</code> object resulting from <code>plot_curve()</code> (or <code>plot_choices()</code> respectively).
<code>plot_b</code>	a <code>ggplot</code> object resulting from <code>plot_choices()</code> (or <code>plot_curve()</code> respectively).
<code>choices</code>	a vector specifying which analytical choices should be plotted. By default, all choices are plotted.
<code>labels</code>	labels for the two parts of the plot
<code>rel_heights</code>	vector indicating the relative heights of the plot.
<code>desc</code>	logical value indicating whether the curve should be arranged in a descending order. Defaults to <code>FALSE</code> .
<code>null</code>	Indicate what value represents the 'null' hypothesis (defaults to zero).

ci	logical value indicating whether confidence intervals should be plotted.
ribbon	logical value indicating whether a ribbon instead should be plotted.
...	additional arguments that can be passed to plot_grid().

Value

a `ggplot` object.

See Also

- `plot_curve()` to plot only the specification curve.
- `plot_choices()` to plot only the choices panel.
- `plot_samplesizes()` to plot a histogram of sample sizes per specification.

Examples

```
# load additional library
library(ggplot2) # for further customization of the plots

# run spec analysis
results <- run_specs(example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1)))

# plot results directly
plot_specs(results)

# Customize each part and then combine
p1 <- plot_curve(results) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "grey") +
  ylim(-3, 12) +
  labs(x = "", y = "regression coefficient")

p2 <- plot_choices(results) +
  labs(x = "specifications (ranked)")

plot_specs(plot_a = p1, # arguments must be called directly!
  plot_b = p2,
  rel_heights = c(2, 2))
```

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "boxplot"`. This function provides a convenient way to visually investigate the effect of individual choices on the estimate of interest. It produces box-and-whisker plot(s) for each provided analytical choice.

Usage

```
plot_summary(df, choices = c("x", "y", "model", "controls", "subsets"))
```

Arguments

<code>df</code>	a data frame resulting from <code>run_specs()</code> .
<code>choices</code>	a vector specifying which analytical choices should be plotted. By default, all choices are plotted.

Value

a [ggplot](#) object.

See Also

[summarise_specs\(\)](#) to investigate the affect of analytical choices in more detail.

Examples

```
# run spec analysis
results <- run_specs(example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1)))

# plot boxplot comparing specific choices
plot_summary(results, choices = c("subsets", "controls", "y"))
```

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can plot a similar figure simply by using the generic `plot()` function and adding the argument `type = "variance"`. This function creates a simple barplot that visually displays how much variance in the outcome (e.g., the regression coefficient) different analytical choices or combinations thereof account for. To use this approach, one needs to estimate a multilevel model that includes all analytical choices as grouping variables (see examples and vignettes). This function uses `icc_specs()` to compute the intraclass correlation coefficients (ICCs), which provides the data basis for the plot (see examples).

Usage

```
plot_variance(model)
```

Arguments

<code>model</code>	a multilevel model that captures the variances of the specification curve (based on the data frame resulting from <code>run_specs</code>).
--------------------	---

Value

a `ggplot` object.

See Also

`icc_specs()` to produce a tibble that details the variance decomposition.

Examples

```
# Step 1: Run spec curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"))

# Step 2: Estimate multilevel model
library(lme4, quietly = TRUE)
model <- lmer(estimate ~ 1 + (1|x) + (1|y), data = results)

# Step 3: Plot model
plot_variance(model)
```

run_specs

*Estimate all specifications***Description**

[Deprecated] This function was deprecated because the new version of `spectr` uses different analytical framework. In this framework, you should use the function `setup()` first and then run all specifications using `spectr()`. This is the central function of the package. It runs the specification curve analysis. It takes the data frame and vectors for analytical choices related to the dependent variable, the independent variable, the type of models that should be estimated, the set of covariates that should be included (none, each individually, and all together), as well as a named list of potential subsets. The function returns a tidy tibble which includes relevant model parameters for each specification. The function `tidy` is used to extract relevant model parameters. Exactly what tidy considers to be a model component varies across models but is usually self-evident.

Usage

```
run_specs(
  df,
  x,
  y,
  model = "lm",
  controls = NULL,
  subsets = NULL,
  all.comb = FALSE,
  conf.level = 0.95,
  keep.results = FALSE
)
```

Arguments

<code>df</code>	a data frame that includes all relevant variables
<code>x</code>	a vector denoting independent variables
<code>y</code>	a vector denoting the dependent variables
<code>model</code>	a vector denoting the model(s) that should be estimated.
<code>controls</code>	a vector denoting which control variables should be included. Defaults to <code>NULL</code> .
<code>subsets</code>	a named list that includes potential subsets that should be evaluated (see examples). Defaults to <code>NULL</code> .
<code>all.comb</code>	a logical value indicating what type of combinations of the control variables should be specified. Defaults to <code>FALSE</code> (i.e., none, all, and each individually). If this argument is set to <code>TRUE</code> , all possible combinations between the control variables are specified (see examples).
<code>conf.level</code>	the confidence level to use for the confidence interval. Must be strictly greater than 0 and less than 1. Defaults to <code>.95</code> , which corresponds to a 95 percent confidence interval.

`keep.results` a logical value indicating whether the complete model object should be kept. Defaults to FALSE.

Value

a [tibble](#) that includes all specifications and a tidy summary of model components.

References

- Simonsohn, U., Simmons, J. P., & Nelson, L. D. (2019). Specification Curve: Descriptive and Inferential Statistics for all Plausible Specifications. Available at: <https://doi.org/10.2139/ssrn.2694998>
- Steegen, S., Tuerlinckx, F., Gelman, A., & Vanpaemel, W. (2016). Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science*, 11(5), 702-712. <https://doi.org/10.1177/1745691616658637>

See Also

[plot_specs\(\)](#) to visualize the results of the specification curve analysis.

Examples

```
# run specification curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"),
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# Check results frame
results
```

setup

Specifying analytical decisions in a specification setup

Description

Creates all possible specifications as a combination of different dependent and independent variables, model types, control variables, potential subset analyses, as well as potentially other analytic choices. This function represents the first step in the analytic framework implemented in the package `spectr`. The resulting class `spectr.setup` then needs to be passed to the core function of the package called `spectr()`, which fits the specified models across all specifications.

Usage

```

setup(
  data,
  x,
  y,
  model,
  controls = NULL,
  subsets = NULL,
  add_to_formula = NULL,
  fun1 = function(x) broom::tidy(x, conf.int = TRUE),
  fun2 = function(x) broom::glance(x),
  simplify = FALSE
)

```

Arguments

<code>data</code>	The data set that should be used for the analysis
<code>x</code>	A character vector denoting the independent variables (which should be numeric variables, see details)
<code>y</code>	A character vector denoting the dependent variables
<code>model</code>	A character vector with the names of the model functions that should be estimated (e.g., "lm" or the name of a custom function), or a named list of functions (e.g., <code>list(ols = lm, robust = MASS::rlm)</code>), in which case the names are used as labels for the models. Any function that accepts the arguments <code>formula</code> and <code>data</code> can be used, including custom wrappers around other model functions (e.g., for survival models, fixed effects models, or weighted regressions; see <code>vignette("different-specifications")</code>).
<code>controls</code>	A vector of the control variables that should be included. Defaults to <code>NULL</code> .
<code>subsets</code>	Specification of potential subsets/groups as list. There are two ways in which these can be specified that both start from the assumption that the "grouping" variable is in the data set. The simplest way is to provide a named vector within the list, whose name is the variable that should be used for subsetting and whose values are the values that reflect the subsets (e.g., <code>list(group2 = c("female", "male"))</code>). In this case, the specifications will include "all", "only female" and "only male". Alternatively, you can also use the <code>unique</code> function to extract that vector directly from the data set (e.g., <code>list(group2 = unique(example_data\$group2))</code>). Both approaches lead to the same result. The former, however, has the advantages that one can also remove some of the subgroups (e.g. <code>list(group2 = c("female"))</code>). In this case, the specifications will include "all" (no subset) and "only females". See examples for more details. Missing values (NA) among the subset values are ignored.
<code>add_to_formula</code>	A string specifying aspects that should always be included in the formula (e.g. a constant covariate, random effect structures...)
<code>fun1</code>	A function that extracts the parameters of interest from the fitted models. Defaults to tidy , which works with a large range of different models.

fun2	A function that extracts fit indices of interest from the models. Defaults to glance , which works with a large range of different models. Note: Different models result in different fit indices. Thus, if you use different models within one specification curve analysis, this may not work. In this case, you can simply set <code>fun2 = NULL</code> to not extract any fit indices.
simplify	Logical value indicating what type of combinations between control variables should be included in the specification. If <code>FALSE</code> (default), all combinations between the provided variables are created (none, each individually, each combination between each variable, all variables). If <code>TRUE</code> , only no covariates, each individually, and all covariates are included as specifications (akin to the default in <code>specr</code> version 0.2.1).

Details

Empirical results are often contingent on analytical decisions that are equally defensible, often arbitrary, and motivated by different reasons. These decisions may introduce bias or at least variability. To this end, specification curve analyses (Simonsohn et al., 2020) or multiverse analyses (Steenen et al., 2016) refer to identifying the set of theoretically justified, statistically valid (and potentially also non-redundant) specifications, fitting the "multiverse" of models represented by these specifications and extract relevant parameters often to display the results graphically as a so-called specification curve. This allows readers to identify consequential specifications decisions and how they affect the results or parameter of interest.

Use of this function

A general overview is provided in the vignette `vignette("getting-started")`. It is assumed that you want to estimate the relationship between two variables (x and y). What varies may be what variables should be used for x and y , what model should be used to estimate the relationship, whether the relationship should be estimated for certain subsets, and whether different combinations of control variables should be included. This allows to (re)produce almost any analytical decision imaginable. See examples below for how a number of typical analytical decision can be implemented. Afterwards you pass the resulting object of a class `specr.setup` to the function `specr()` to run the specification curve analysis.

As `specr()` extracts the estimate whose term equals x , the independent variables should be numeric. For factor or character variables, models return terms such as " x_B ", so recode them as (dummy) variables instead.

Note, the resulting class of `specr.setup` allows to use generic functions. Use `methods(class = "specr.setup")` for an overview on available methods and e.g., `?summary.specr.setup` to view the dedicated help page.

Value

An object of class `specr.setup` which includes all possible specifications based on combinations of the analytic choices. The resulting list includes a specification tibble, the data set, and additional information about the universe of specifications. Use `methods(class = "specr.setup")` for an overview on available methods.

References

- Simonsohn, U., Simmons, J.P. & Nelson, L.D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. <https://doi.org/10.1038/s41562-020-0912-z>
- Steegen, S., Tuerlinckx, F., Gelman, A., & Vanpaemel, W. (2016). Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science*, 11(5), 702-712. <https://doi.org/10.1177/174569>

See Also

`spectr()` for the second step of actually running the actual specification curve analysis
`summary.spectr.setup()` for how to summarize and inspect the resulting specifications
`plot.spectr.setup()` for creating a visual summary of the specification setup.

Examples

```
## Example 1 ----
# Setting up typical specifications
specs <- setup(data = example_data,
  x = c("x1", "x2"),
  y = c("y1", "y2"),
  model = "lm",
  controls = c("c1", "c2", "c3"),
  subsets = list(group1 = c("young", "middle", "old"),
    group2 = c("female", "male")),
  simplify = TRUE)

# Check specifications
summary(specs, rows = 18)

## Example 2 ----
# Setting up specifications for multilevel models
specs <- setup(data = example_data,
  x = c("x1", "x2"),
  y = c("y1", "y2"),
  model = c("lmer"), # multilevel model
  subsets = list(group1 = c("young", "old"), # only young and old!
    group2 = unique(example_data$group2)), # alternative specification
  controls = c("c1", "c2"),
  add_to_formula = "(1|group2)") # random effect in all models

# Check specifications
summary(specs)

## Example 3 ----
# Setting up specifications with a different parameter extract functions

# Create custom extract function to extract different parameter and model
tidy_99 <- function(x) {
  fit <- broom::tidy(x,
```

```

      conf.int = TRUE,
      conf.level = .99)      # different alpha error rate
  fit$full_model = list(x)    # include entire model fit object as list
  return(fit)
}

# Setup specs
specs <- setup(data = example_data,
  x = c("x1", "x2"),
  y = c("y1", "y2"),
  model = "lm",
  fun1 = tidy_99,            # pass new function to setup
  add_to_formula = "c1 + c2") # set of covariates in all models

# Check specifications
summary(specs)

## Example 4 ----
# Model functions as a named list (the names are used as labels)

# Custom function, e.g. a weighted regression (any function with the
# arguments `formula` and `data` works, e.g. also wrappers around
# fixest::feols() or survival::coxph(), see vignette("different-specifications"))
weighted_lm <- function(formula, data) {
  lm(formula, data = data, weights = rep(1:2, length.out = nrow(data)))
}

specs <- setup(data = example_data,
  x = c("x1", "x2"),
  y = "y1",
  model = list(ols = lm, weighted = weighted_lm),
  controls = "c1")

# Check specifications
summary(specs)

```

specr

Fit models across all specifications

Description

Runs the specification/multiverse analysis across specified models. This is the central function of the package and represent the second step in the analytic framework implemented in the package specr. It estimates and returns respective parameters and estimates of models that were specified via setup().

Usage

```
specr(x, data = NULL, ...)
```

Arguments

<code>x</code>	A <code>specr.setup</code> object resulting from <code>setup</code> or a tibble that contains the relevant specifications (e.g., a tibble resulting from <code>as_tibble(setup(...))</code>).
<code>data</code>	If <code>x</code> is not an object of "specr.setup" and simply a tibble, you need to provide the data set that should be used. Defaults to <code>NULL</code> as it is assumed that most users will create an object of class "specr.setup" that they'll pass to <code>specr()</code> .
<code>...</code>	Further arguments that can be passed to <code>future_pmap</code> . This only becomes important if parallelization is used (they are ignored if the specifications are fitted sequentially). For example, <code>furrr_options</code> can be passed to the argument <code>.options</code> . When a plan for parallelization is set, one can also set <code>.progress = TRUE</code> to print a progress bar during the fitting process. See details for more information on parallelization.

Details

Empirical results are often contingent on analytical decisions that are equally defensible, often arbitrary, and motivated by different reasons. These decisions may introduce bias or at least variability. To this end, specification curve analyses (Simonsohn et al., 2020) or multiverse analyses (Steenen et al., 2016) refer to identifying the set of theoretically justified, statistically valid (and potentially also non-redundant) specifications, fitting the "multiverse" of models represented by these specifications and extract relevant parameters often to display the results graphically as a so-called specification curve. This allows readers to identify consequential specifications decisions and how they affect the results or parameter of interest.

Use of this function

A general overview is provided in the vignette `vignette("getting-started")`. Generally, you create relevant specification using the function `setup()`. You then pass the resulting object of a class `specr.setup` to the present function `specr()` to run the specification curve analysis. Further note that the resulting object of class `specr.object` allows to use several generic function such as `summary()` or `plot()`. Use `methods(class = "specr.object")` for an overview on available methods and e.g., `?plot.specr.object` to view the dedicated help page.

Parallelization

By default, the function fits models across all specifications sequentially (one after the other). If the data set is large, the models complex (e.g., large structural equation models, negative binomial models, or Bayesian models), and the number of specifications is large, it can make sense to parallelize these operations. One simply has to load the package `furrr` (which in turn, builds on `future`) up front. Then parallelizing the fitting process works as specified in the package description of `furrr/future` by setting a "plan" before running `specr` such as:

```
plan(multisession, workers = 4)
```

However, there are many more ways to specifically set up the plan, including different strategy than `multisession`. For more information, see `vignette("parallelization")` and the [reference page](#) for `plan()`.

Disclaimer

We do see a lot of value in investigating how analytical choices affect a statistical outcome of interest. However, we strongly caution against using `specr` as a tool to somehow arrive at a better estimate compared to a single model. Running a specification curve analysis does not make your

findings any more reliable, valid or generalizable than a single analysis. The method is meant to inform about the effects of analytical choices on results, and not a better way to estimate a correlation or effect.

Value

An object of class `specr.object`, which includes a data frame with all specifications their respective results along with many other useful information about the model. Parameters are extracted via the function passed to `setup`. By default this is `broom::tidy()` and the function `broom::glance()`. Several other aspects and information are included in the resulting class (e.g., number of specifications, time elapsed, subsets included in the analyses). Use `methods(class = "specr.object")` for an overview on available methods.

References

- Simonsohn, U., Simmons, J.P. & Nelson, L.D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. <https://doi.org/10.1038/s41562-020-0912-z>
- Steegen, S., Tuerlinckx, F., Gelman, A., & Vanpaemel, W. (2016). Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science*, 11(5), 702-712. <https://doi.org/10.1177/174569>

See Also

[setup\(\)](#) for the first step of setting up the specifications.

[summary.specr.object\(\)](#) for how to summarize and inspect the results.

[plot.specr.object\(\)](#) for plotting results.

Examples

```
# Example 1 ----
# Setup up typical specifications
specs <- setup(data = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1)))
```

```
# Run analysis (not parallelized)
results <- specr(specs)
```

```
# Summary of the results
summary(results)
```

```
# Example 2 ----
# Working without S3 classes
specs2 <- setup(data = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
```

```

controls = "c1")

# Working with tibbles
specs_tibble <- as_tibble(specs2)      # extract tibble from setup
results2 <- specr(specs_tibble,
                  data = example_data) # need to provide data!

# Results (tibble instead of S3 class)
head(results2)

```

summarise_specs	<i>Summarise specifications</i>
-----------------	---------------------------------

Description

[Deprecated] This function is deprecated because the new version of `specr` uses a new analytic framework. In this framework, you can obtain a similar summary simply by using the generic `summary()` function and adding the argument `type = "curve"`. This function allows to inspect results of the specification curves by returning a comparatively simple summary of the results. This summary can be produced for various specific analytical choices and customized summary functions.

Usage

```

summarise_specs(
  df,
  ...,
  var = .data$estimate,
  stats = list(median = median, mad = mad, min = min, max = max, q25 = function(x)
    quantile(x, prob = 0.25), q75 = function(x) quantile(x, prob = 0.75))
)

```

Arguments

<code>df</code>	a data frame resulting from <code>run_specs()</code> .
<code>...</code>	one or more grouping variables (e.g., subsets, controls,...) that denote the available analytical choices.
<code>var</code>	which variable should be evaluated? Defaults to <code>estimate</code> (the effect sizes computed by <code>run_specs()</code>).
<code>stats</code>	named vector or named list of summary functions (individually defined summary functions can be included). If it is not named, placeholders (e.g., "fn1") will be used as column names.

Value

a [tibble](#).

See Also

[plot_summary\(\)](#) to visually investigate the affect of analytical choices.

Examples

```
# Run specification curve analysis
results <- run_specs(df = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = c("lm"),
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1),
    group2 = unique(example_data$group2)))

# overall summary
summarise_specs(results)

# Summary of specific analytical choices
summarise_specs(results,      # data frame
  x, y)                      # analytical choices

# Summary of other parameters across several analytical choices
summarise_specs(results,
  subsets, controls,
  var = p.value,
  stats = list(median = median,
    min = min,
    max = max))

# Named vector instead of named list passed to `stats`
summarise_specs(results,
  controls,
  stats = c(mean = mean,
    median = median))
```

summary.spectr.infer	<i>Summarizing inference on the specification curve</i>
----------------------	---

Description

Computes the three test statistics proposed by Simonsohn et al. (2020) for the observed specification curve and for each curve under the null (resulting from [infer_curve\(\)](#)), and tests how extreme the observed statistics are compared to their distribution under the null.

Usage

```
## S3 method for class 'spectr.infer'
summary(
```

```

object,
direction = c("two.sided", "greater", "less"),
alpha = 0.05,
group = NULL,
...
)

```

Arguments

<code>object</code>	A <code>spectr.infer</code> object, resulting from calling <code>infer_curve()</code> .
<code>direction</code>	Direction of the alternative hypothesis. The default "two.sided" tests for an effect in either direction. Use "greater" or "less" only if the direction of the effect was predicted before the results were known.
<code>alpha</code>	Significance level that is used to count significant specifications. Defaults to .05.
<code>group</code>	Optional character vector with names of analytical choices (e.g., "y" or "subsets") to compute the test statistics separately within each group of specifications.
<code>...</code>	Further arguments passed to or from other methods (currently ignored).

Details

The three test statistics are:

median The median estimate across all specifications.

share significant The number of specifications with a significant estimate ($p < \alpha$) in the tested direction: positive for "greater", negative for "less", and the more frequent (dominant) direction for "two.sided".

aggregated z Stouffer's Z: each p-value is converted into a z-value with the sign of its estimate, and the sum of these z-values is divided by the square root of the number of specifications.

The p-value of each statistic is the share of curves under the null whose statistic is at least as extreme as the observed one, computed as $(1 + \text{number of curves at least as extreme}) / (1 + \text{number of bootstrap samples})$, so that it is never zero. For "two.sided", "at least as extreme" refers to the absolute value of the median and of Stouffer's Z, and to the number of significant specifications in the dominant direction of each curve. If the extraction function did not return p-values, only the median can be tested.

Value

A tibble with one row per test statistic (and group), including the observed estimate, the 2.5% and 97.5% quantiles of the statistic under the null (`null_lower` and `null_upper`), the p.value, and the number of specifications (`n_specs`).

References

- Simonsohn, U., Simmons, J.P. & Nelson, L.D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. <https://doi.org/10.1038/s41562-020-0912-z>

See Also

[infer_curve\(\)](#), [plot.speccr.infer\(\)](#)

Examples

```
specs <- setup(data = example_data,
  y = "y1",
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"))

results <- speccr(specs)

set.seed(42)
inference <- infer_curve(results, specs, n_samples = 200) # better 1,000 or more

# Two-sided tests (default)
summary(inference)

# One-sided tests for a predicted positive effect
summary(inference, direction = "greater")

# Separately for each independent variable
summary(inference, group = "x")
```

summary.speccr.object *Summarizing the Specification Curve Analysis*

Description

summary method for class "speccr.object". With type = "default", it provides a printed output including technical details (e.g., cores used, duration of the fitting process, number of specifications), a descriptive analysis of the overall specification curve, a descriptive summary of the resulting sample sizes, and a head of the results. With type = "curve", it returns descriptive statistics of the specification curve (overall or for groups of specifications).

Usage

```
## S3 method for class 'speccr.object'
summary(
  object,
  type = "default",
  group = NULL,
  var = .data$estimate,
  stats = list(median = median, mad = mad, min = min, max = max, q25 = function(x)
    quantile(x, prob = 0.25), q75 = function(x) quantile(x, prob = 0.75)),
  digits = 2,
  rows = 6,
```

```
    ...  
  )
```

Arguments

object	An object of class "speccr.object", usually resulting of a call to speccr.
type	What should be summarized? Either "default" (printed summary of the whole analysis) or "curve" (tibble with descriptive statistics of the curve).
group	In combination with type = "curve", a character vector of one or more variables (e.g., "subsets", "controls",...) that denote the available analytic choices to group the summary of the estimate.
var	Unquoted name of the parameter to be summarized. Defaults to estimate.
stats	Named vector or named list of summary functions (individually defined summary functions can included). If it is not named, placeholders (e.g., "fn1") will be used as column names.
digits	The number of digits to use when printing the specification table.
rows	The number of rows of the specification tibble that should be printed.
...	further arguments passed to or from other methods (currently ignored).

Value

For type = "default", a printed summary of an object of class speccr.object, which is returned invisibly. For type = "curve", a tibble with the descriptive statistics (and the median sample size obs, if available).

See Also

The function used to create the "speccr.object" object: [speccr\(\)](#).

Examples

```
# Setup up specifications (returns object of class "speccr.setup")
specs <- setup(data = example_data,
  y = c("y1", "y2"),
  x = c("x1", "x2"),
  model = "lm",
  controls = c("c1", "c2"),
  subsets = list(group1 = unique(example_data$group1)))

# Run analysis (returns object of class "speccr.object")
results <- speccr(specs)

# Default summary of the "speccr.object"
summary(results)

# Summarize the specification curve descriptively
summary(results, type = "curve")

# Grouping for certain analytical decisions
```

```
summary(results,
        type = "curve",
        group = c("x", "y"))

# Using customized functions
summary(results,
        type = "curve",
        group = c("x", "group1"),
        stats = list(median = median,
                     min = min,
                     max = max))
```

summary.spcr.setup	<i>Summarizing the Specifications Setup</i>
--------------------	---

Description

summary method for class "spcr.setup". Provides a short summary of the created specifications (the "multiverse") that lists all analytic choices, prints the function used to extract the parameters from the model. Finally, if print.specs = TRUE, it also shows the head of the actual specification grid.

Usage

```
## S3 method for class 'spcr.setup'
summary(object, digits = 2, rows = 6, print.specs = TRUE, ...)
```

Arguments

object	An object of class "spcr.setup", usually, a result of a call to setup.
digits	The number of digits to use when printing the specification table.
rows	The number of rows of the specification tibble that should be printed.
print.specs	Logical value; if TRUE, the head of the specification tibble is printed.
...	further arguments passed to or from other methods (currently ignored).

Value

Prints a summary of an object of class spcr . setup and returns the object invisibly.

See Also

The function [setup\(\)](#), which creates the "spcr.setup" object.

Examples

```
# Setup specifications
specs <- setup(data = example_data,
  x = c("x1", "x2"),
  y = c("y1", "y2"),
  model = c("lm", "glm"),
  controls = c("c1", "c2", "c3"),
  subsets = list(group3 = unique(example_data$group3)))

# Summarize specifications
summary(specs)
```

Index

- * **datasets**
 - example_data, [2](#)
- example_data, [2](#)
- furrr::future_map(), [4](#)
- ggplot, [7](#), [9](#), [11–19](#)
- glance, [23](#)
- icc_specs, [3](#)
- icc_specs(), [19](#)
- infer_curve, [4](#)
- infer_curve(), [6](#), [7](#), [29–31](#)
- plot(), [4](#)
- plot.spcr.infer, [6](#)
- plot.spcr.infer(), [5](#), [31](#)
- plot.spcr.object, [7](#)
- plot.spcr.object(), [27](#)
- plot.spcr.setup, [10](#)
- plot.spcr.setup(), [24](#)
- plot_choices, [11](#)
- plot_choices(), [17](#)
- plot_curve, [12](#)
- plot_curve(), [17](#)
- plot_decisiontree, [14](#)
- plot_samplesizes, [15](#)
- plot_samplesizes(), [17](#)
- plot_specs, [16](#)
- plot_specs(), [21](#)
- plot_summary, [17](#)
- plot_summary(), [29](#)
- plot_variance, [18](#)
- plot_variance(), [3](#)
- run_specs, [20](#)
- run_specs(), [12–15](#), [28](#)
- set.seed(), [5](#)
- setup, [21](#)
- setup(), [4](#), [20](#), [27](#), [33](#)
- specr, [25](#)
- specr(), [4](#), [20](#), [24](#), [32](#)
- summarise_specs, [28](#)
- summarise_specs(), [18](#)
- summary(), [4](#)
- summary.spcr.infer, [29](#)
- summary.spcr.infer(), [5](#), [7](#)
- summary.spcr.object, [31](#)
- summary.spcr.object(), [27](#)
- summary.spcr.setup, [33](#)
- summary.spcr.setup(), [24](#)
- tibble, [16](#), [21](#), [28](#)
- tidy, [20](#), [22](#)