Package 'specr'

May 9, 2026

Title Conducting and Visualizing Specification Curve Analyses

Version 1.0.0

Description Provides utilities for conducting specification curve analyses (Simonsohn, Simmons & Nelson (2020, <doi:10.1038/s41562-020-0912-z>) or multiverse analyses (Steegen, Tuerlinckx, Gelman & Vanpaemel, 2016, <doi:10.1177/1745691616658637>) including functions to setup, run, evaluate, and plot all specifications.

License GPL-3

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

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

Depends R (>= 3.5.0)

Imports broom, cowplot, dplyr, furrr, future, ggplot2, ggraph, glue, igraph, lifecycle, lme4, magrittr, methods, parallelly, purrr, rlang, stringr, tibble, tidyr

Suggests broom.mixed, gapminder, ggridges, knitr, lavaan, testthat, tidyverse, performance, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

VignetteBuilder knitr

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 2023-01-20 13:50:02 UTC

Contents

`example_data` *Example data set*

Description

This simulated data set can be used to explore the major function of 'specr'. 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

Examples

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

icc_specs *Compute intraclass correlation coefficient*

Description

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 therefor 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

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

Value

a tibble 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)
```

`plot.specr.object` *Plot specification curve and analytic choices*

Description

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

Usage

```
## S3 method for class 'specr.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,
  ...
)
```

Arguments

`x`	A `specr.object` object, usually resulting from calling `specr()`.
`type`	What type of figure should be plotted? If `type = "default"`, 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 `type = "curve"`, only the specification curve (upper panel of the default plot) is plotted. If `type = "choices"`, only the choice panel (lower part of the default plot) is plotted. If `type = "boxplot"`, an alternative visualization of differences between choices is plotted that summarizes results per choice using box-and-whisker plot(s). If `type = "samplesizes"`, a barplot of sample sizes per specification is plotted. See examples for more information.
`var`	Which parameter should be plotted in the curve? Defaults to `estimate`, but other parameters (e.g., p.value, fit_r.squared,...) can be plotted too.

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 analytic choices should be plotted. By default, all choices (x, y, model, controls, subsets) are plotted.
labels	Labels for the two parts of the plot
rel_heights	vector indicating the relative heights of the plot.
desc	Logical value indicating whether the curve should the arranged in a descending order. Defaults to FALSE.
null	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
formula	In combination with `type = "variance"`, you can provide a specific formula to extract specific variance components. The syntax of the formula is based on `lme4::lmer()` and thus looks something like, e.g.: `"estimate ~ 1 + (1|x) + (1|y)"` (to estimate the amount of variance explained by different independent `x` and dependent variables `y`). All other choices are then subsumed under residual variance. By 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 analstical choices) should be printed or not. Defaults to TRUE.
...	further arguments passed to or from other methods (currently ignored).

Value

A ggplot object that can be customized further.

Examples

```
## Not run:
# 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 <- specr(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")


# 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 chocies
plot(results,
     type = "variance",
     formula = "estimate ~ 1 + (1|x) + (1|y) + (1|group1) + (1|x:y)")

## Combining several plots ----
# `specr` 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)

## End(Not run)
```

`plot.specr.setup` *Plot visualization of the specification setup*

Description

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

Usage

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

Arguments

`x`	A `specr.setup` object, usually resulting from calling `setup()`.
`layout`	The type of layout to create for the garden of forking path. Defaults to "dendrogram". See `?ggraph` 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

```
## Not run:
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)

## End(Not run)
```

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 the 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 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 = "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 `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...)
`desc`	logical value indicating whether the curve should the 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.
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 *Plot decision tree*

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()`. 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

df	data frame resulting from run_specs().
label	Logical. Should labels be included? Defaults to FALSE. Produces only a reasonable plot if number of specifications is low.
legend	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 *Plot sample sizes*

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

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...)
desc	logical value indicating whether the curve should the arranged in a descending order. Defaults to FALSE.

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 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 = "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

df	a data frame resulting from `run_specs()`.
plot_a	a ggplot object resulting from `plot_curve()` (or `plot_choices()` respectively).
plot_b	a ggplot object resulting from `plot_choices()` (or `plot_curve()` respectively).
choices	a vector specifying which analytical choices should be plotted. By default, all choices are plotted.
labels	labels for the two parts of the plot
rel_heights	vector indicating the relative heights of the plot.
desc	logical value indicating whether the curve should the arranged in a descending order. Defaults to FALSE.
null	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"),
                     subset = 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_height = c(2, 2))
```

---

plot_summary *Create box plots for given analytical choices*

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

`df`	a data frame resulting from `run_specs()`.
`choices`	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"),
                     subset = list(group1 = unique(example_data$group1)))

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

---

plot_variance *Plot variance decomposition*

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 functions creates a simple barplot that visually displays how much variance in the outcome (e.g., the regression coefficient) different analytical choices or combinations therefor 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

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

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 specr uses different analytical framework. In this framework, you should use the function setup() first and then run all specifications using specr(). 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

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

`data`	The data set that should be used for the analysis
`x`	A vector denoting independent variables
`y`	A vector denoting the dependent variables
`model`	A vector denoting the model(s) that should be estimated.
`controls`	A vector of the control variables that should be included. Defaults to NULL.
`subsets`	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., `list(group2 = c("female", "male"))`. In this case, the specifications will includes "all", "only female" and "only male". Alternatively, you can also use the `unique` function to extract that vector directly from the data set (e.g., `list(group2 = unique(example_data$group2)`. Both approaches lead to the same result. The former, however, has the advantages that one can also remove some of the subgroups (e.g. `list(group2 = c("female"))`. In this case, the specifications will include "all" (no subset) and "only females". See examples for more details.

`add_to_formula` A string specifying aspects that should always be included in the formula (e.g. a constant covariate, random effect structures...)

`fun1` 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 `fun2 = NULL` to not extract any fit indices.

`simplify` Logical value indicating what type of combinations between control variables should be included in the specification. If FALSE (default), all combinations between the provided variables are created (none, each individually, each combination between each variable, all variables). If TRUE, only no covariates, each individually, and all covariates are included as specifications (akin to the default in specr version 0.2.1).

Details

Empirical results are often contingent on analytical decisions that are equally defensible, often arbitrary, and motivated by different reasons. This decisions may introduce bias or at least variability. To this end, specification curve analyses (Simonsohn et al., 2020) or multiverse analyses (Steegen 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 vignettes `vignette("specr")`. 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.

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

`specr()` for the second step of actually running the actual specification curve analysis

`summary.specr.setup()` for how to summarize and inspect the resulting specifications

`plot.specr.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)
```

---

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

`x`	A `specr.setup` object resulting from `setup` or a tibble that contains the relevant specifications (e.g., a tibble resulting from `as_tibble(setup(...))`).
`data`	If x is not an object of "specr.setup" and simply a tibble, you need to provide the data set that should be used. Defaults to NULL as it is assumend that most users will create an object of class "specr.setup" that they'll pass to `specr()`.
`...`	Further arguments that can be passed to `future_pmap`. This only becomes important if parallelization is used. For example, if a custom model function is used this involves passing `furrr_options` passing to the argument `.options`. When a plan for parallelization is set, one can also set `.progress = TRUE` 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. This decisions may introduce bias or at least variability. To this end, specification curve analyses (Simonsohn et al., 2020) or multiverse analyses (Steegen 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 vignettes `vignette("specr")`. 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 `furr/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 *Summarise specifications*

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. 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

`df`	a data frame resulting from `run_specs()`.
`...`	one or more grouping variables (e.g., subsets, controls,...) that denote the available analytical choices.
`var`	which variable should be evaluated? Defaults to estimate (the effect sizes computed by `run_specs()`).
`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.

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))

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

`summary.specr.object` *Summarizing the Specification Curve Analysis*

Description

`summary` method for class "specr". 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.

Usage

```
## S3 method for class 'specr.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 "specr", usually resulting of a call to specr.
type	Different aspects can be summarized and printed. See details for alternative summaries
group	In combination with what = "curve", provide a vector of one or more variables (e.g., subsets, controls,...) that denote the available analytic choices to group summary of the estimate.
var	In combination with what = "curve", unquoted name of 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

A printed summary of an object of class specr.object.

See Also

The function used to create the "specr.setup" object: setup.

Examples

```
# Setup up specifications (returns object of class "specr.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 "specr.object")
results <- specr(specs)

# Default summary of the "specr.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.specr.setup` *Summarizing the Specifications Setup*

Description

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

Arguments

`object`	An object of class "specr.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`, a head of the specification tibble is returned and printed.
`...`	further arguments passed to or from other methods (currently ignored).

Value

A printed summary of an object of class `specr.setup`.

See Also

The function `setup()`, which creates the "specr.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