Package 'specr'

March 26, 2020

Title Conducting and Visualizing Specification Curve Analyses

Version 0.2.1

Description Provides utilities for conducting specification curve analyses (Simonsohn, Simmons & Nelson (2015, <doi: 10.2139/ssrn.2694998>) 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, ggplot2, ggraph, glue, igraph, lme4, magrittr, purrr, rlang, tibble, tidyr

Suggests knitr, testthat, tidyverse, performance, progress, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2020-03-26 13:40:02 UTC

R topics documented:

`example_data` *Example data set*

Description

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

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_choices *Plot how analytical choices affect results*

Description

This functions plots how analytical 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,
  choices = c("x", "y", "model", "controls", "subsets"),
  desc = FALSE,
  null = 0
)
```

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.
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

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,
  desc = FALSE,
  ci = TRUE,
  ribbon = FALSE,
  legend = FALSE,
  null = 0
)
```

Arguments

df	a data frame resulting from `run_specs()`.
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

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

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, desc = FALSE)
```

Arguments

`df`	a data frame resulting from `run_specs()`.
`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)

# customize
plot_samplesizes(results) +
  geom_hline(yintercept = median(results$obs),
             color = "darkgrey",
             linetype = "dashed") +
  theme_linedraw()
```

plot_specs *Plot specification curve and analytical choices*

Description

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,
  sample_perc = 1,
  ...
)
```

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.

sample_perc	numeric value denoting what percentage of the specifications should be plotted. Needs to be strictly greater than 0 and smalle than 1. Defaults to 1 (= all specifications). Drawing a sample from all specification usually makes only sense of the number of specifications is very large and one wants to simplify the visualization.
...	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

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

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

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,
  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.
`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_specs` *Set up specifications*

Description

This function creates a tibble that includes all possible specifications based the dependent and independent variables, model types, and control variables that are specified. This function simply produces a tibble of all combinations. It can be used to check the specified analytical choices. This function is called within `run_specs()`, which estimates all specified models based on the data that are provided.

Usage

```
setup_specs(x, y, model, controls = NULL)
```

Arguments

`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.

Value

a tibble that includes all possible specifications based on combinations of the analytical choices.

See Also

`run_specs()` to run the specification curve analysis.

Examples

```
setup_specs(y = c("y1"),
            x = c("x1", "x2"),
            model = c("lm"),
            controls = c("c1", "c2"))
```

`summarise_specs` *Summarise specifications*

Description

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, median))
```

Index