Package 'procs'

March 20, 2026

Type Package

Title Recreates Some 'SAS®' Procedures in 'R'

Version 1.0.8

Maintainer David Bosak <dbosak01@gmail.com>

Description Contains functions to simulate the most commonly used
'SAS®' procedures. Specifically, the package aims to
simulate the functionality of 'proc freq', 'proc means', 'proc ttest',
'proc reg', 'proc transpose', 'proc sort', and 'proc print'.
The simulation will include recreating
all statistics with the highest fidelity possible.

License CC0

Encoding UTF-8

URL https://procs.r-sassy.org, https://github.com/dbosak01/procs

BugReports https://github.com/dbosak01/procs/issues

Depends R (>= 3.6.0), common (>= 1.1.5)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), covr, logr, ggplot2

Imports utils, fmtr, reporter, stats, tibble, sasLM (>= 0.9.9),
graphics, withr

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author David Bosak [aut, cre],
Diyu Yang [aut],
Athenkosi Nkonyeni [aut],
Kevin Kramer [ctb],
Brian Varney [ctb],
Duong Tran [ctb],
Yifei Chen [ctb]

Repository CRAN

Date/Publication 2026-03-20 06:10:24 UTC

Contents

`freqplot` *Request a Frequency Plot*

Description

A function to request a frequency plot on a call to `proc_freq`. The function allows you to specify the type of frequency plot to produce, and various layout options. It supports bar charts and dot plots for one and two-way analysis. It also supports vertical or horizontal orientation, by variables, and various scale options.

Usage

```
freqplot(
  type = "barchart",
  orient = "vertical",
  scale = "freq",
  twoway = "groupvertical",
  groupby = "column",
  npanelpos = 4
)
```

Arguments

`type`	A string indicating the type of plot to create. Valid values are "barchart" or "dotplot". Default is "barchart".
`orient`	The orientation of the plot. Valid values are "vertical" or "horizontal". Default is "vertical".
`scale`	The scale to use for the plot. Valid values are "freq", "log", "percent", or "sqrt". Default is "freq". For two-way tables, the value "grouppercent" is also valid.
`twoway`	Options for two-way layouts. Valid values are "cluster", "grouphorizontal", "groupvertical", or "stacked". Default is "groupvertical".

groupby	The variable configuration for two-way charts. Valid values are "column" or "row". Default is "column", which means the first variable in the interaction will be used for the "column" variable, and the second variable for the "rows" on the panel. The "row" option effectively reverses the variable configuration.
npanelpos	The number of charts per panel. Default is 4. This parameter applies to two-way tables only.

Details

The freqplot function can be passed to the plots parameter on proc_freq to give you some control over what kind of frequency plot is produced. Generally, one call to freqplot produces one frequency plot. The exception is on two-way interactions when there are more interactions that can fit on a single panel. In this case, the function will produce multiple panels so that frequencies for all interactions are displayed. The number of plots on a panel can be controlled using the npanelpos parameter.

The default plot is a simple bar chart showing frequency counts for each category. The X axis will show the categories, and the Y axis will show the frequency counts. You may change the orientation of the chart using the orient parameter. You may also change the Y scale using the scale parameter. Options include a percentage, logarithmic, or square root scale.

There are more options for two-way interactions. For two-way interactions, you may choose to display the data as stacked bar charts or clustered bar charts. To create stacked or clustered bar charts, set the type parameter to "barchart", and then specify the desired layout with the twoway parameter.

The "grouphorizontal" and "groupvertical" options on the twoway parameter apply to both bar charts and dot plots. These keywords control whether you want the frequency groups displayed horizontally across the panel, or vertically from top to bottom. The default is to display groups vertically.

The layout for two-way charts can be further manipulated with the groupby parameter. Normally for two-way charts the first variable is the X axis "columns" and the second variable is the grouping variable "rows". Passing a value of "rows" to the groupby parameter essentially flips this orientation, so that the first variable becomes the rows and the second variable becomes the columns. Note that these types of charts can be manipulated further with the orient parameter.

By combining the above options, you can produce many styles of frequency plots.

Any requested plots will be displayed on the interactive report only. Plots are created as jpeg files, and stored in a temp directory. Those temporary files are then referenced by the interactive report to display the graphic.

If desired, you may output the report objects and pass to proc_print. To do this, set output = report on the call to proc_freq, and pass the resulting list to proc_print.

Value

The frequency plot object. This object is then passed to proc_freq for evaluation and rendering. Data for the frequency plot comes directly from the proc_freq reporting data frames.

See Also

proc_freq()

Examples

```
library(procs)

# Turn off printing for CRAN checks
# Set to TRUE to run in local environment
options("procs.print" = FALSE)

# Prepare sample data
dt <- as.data.frame(HairEyeColor, stringsAsFactors = FALSE)

# Example 1: Single plot request for all tables
res <- proc_freq(dt, tables = v(Hair, Eye, Hair * Eye),
                 weight = Freq,
                 plots = freqplot(type = "dotplot"),
                 titles = "Hair and Eye Frequency Statistics")

# View results
res

# Example 2: Separate plots request for each table
res <- proc_freq(dt, tables = v(Hair, Eye, Hair * Eye),
                 weight = Freq,
                 plots = list(freqplot(type = "barchart",
                                       orient = "horizontal"),
                              freqplot(type = "dotplot",
                                       scale = "percent"),
                              freqplot(type = "barchart",
                                       twoway = "cluster")),
                 titles = "Hair and Eye Frequency Statistics")

# View results
res

# Example 3: Display options for group orientation
res <- proc_freq(dt, tables = v(Hair * Eye, Eye * Hair),
                 weight = Freq,
                 plots = list(freqplot(type = "dotplot",
                                       twoway = "grouphorizontal"),
                              freqplot(type = "dotplot",
                                       twoway = "groupvertical")),
                 titles = "Hair and Eye Frequency Statistics")

# View results
res
```

---

proc_freq *Generates Frequency Statistics*

Description

The proc_freq function generates frequency statistics. It is both an interactive function that can be used for data exploration, and can produce dataset output for further analysis. The function can perform one and two-way frequencies. Two-way frequencies are produced as a cross-tabulation by default. There are many options to control the generated tables. The function will return requested tables in a named list.

Usage

```
proc_freq(
  data,
  tables = NULL,
  output = NULL,
  by = NULL,
  weight = NULL,
  options = NULL,
  titles = NULL,
  order = "internal",
  plots = NULL,
  where = NULL
)
```

Arguments

data	The input data frame to perform frequency calculations on. Input data as the first parameter makes this function pipe-friendly.
tables	The variable or variables to perform frequency counts on. The table specifications are passed as a vector of strings. For one-way frequencies, simply pass the variable name. For two-way tables, pass the desired combination of variables separated by a star (*) operator. The parameter does not accept SAS® style grouping syntax. All cross combinations should be listed explicitly. If the table request is named, the name will be used as the list item name on the return list of tables. See "Example 3" for an illustration on how to name an output table.
output	Whether or not to return datasets from the function. Valid values are "out", "none", and "report". Default is "out". This parameter also accepts the data shaping options "long", "stacked", and "wide". See the **Data Shaping** section for a description of these options. Multiple output keywords may be passed on a character vector. For example, to produce both a report dataset and a "long" output dataset, use the parameter `output = c("report", "out", "long")`.
by	An optional by group. Parameter accepts a vector of one or more variable names. When this parameter is set, data will be subset for each by group, and tables will be generated for each subset.
weight	An optional weight parameter. This parameter is passed as a variable name to use for the weight. If a weight variable is indicated, the weighted value will be summed to calculate the frequency counts.
options	The options desired for the function. Options are passed to the parameter as a vector of quoted strings. You may also use the `v()` function to pass unquoted

strings. The following options are available: "chisq", "crosstab", "fisher", "list", "missing", "nlevels", "nocol", "nocum", "nofreq", "nopercent", "noprint", "nonobs", "norow", "nosparse", "notable", "outcum". See the **Options** section for a description of these options.

`titles` A vector of titles to assign to the interactive report.

`order` Indicates how to order the function output. Options are "internal", "formatted", "freq" and "data". Default is "internal". See the section on "Ordering Frequency Output" for more information on this parameter.

`plots` Pass the desired plot(s) on this parameter. Valid values are TRUE, NULL, or a call to the `freqplot` function. Default is NULL, meaning no plots are desired. The value TRUE will give a default frequency plot for each table request. If there are multiple table requests, you can pass a single plot request which will apply to all tables, or a list of plot requests that align one-to-one for each table request.

`where` An expression to filter the rows before the frequencies are calculated. Use the `expression` function to define the filter.

Details

The `proc_freq` function generates frequency statistics for one-way and two-way tables. Data is passed in on the `data` parameter. The desired frequencies are specified on the `tables` parameter.

Value

The function will return all requested datasets by default. This is equivalent to the `output = "out"` option. To return the datasets as created for the interactive report, pass the "report" output option. If no output datasets are desired, pass the "none" output option. If a single dataset is requested, the function will return a single dataset. If multiple datasets are requested, the function will return a list of datasets. The type of data frame returned will correspond to the type of data frame passed in on the `data` parameter. If the input data is a tibble, the output data will be a tibble. If the input data is a Base R data frame, the output data will be a Base R data frame.

Report Output

By default, `proc_freq` results will be immediately sent to the viewer as an HTML report. This functionality makes it easy to get a quick analysis of your data with very little effort. To turn off the interactive report, pass the "noprint" keyword to the `options` parameter or set `options("procs.print" = FALSE)`.

The `titles` parameter allows you to set one or more titles for your report. Pass these titles as a vector of strings.

If the frequency variables have a label assigned, that label will be used in the report output. This feature gives you some control over the column headers in the final report.

The exact datasets used for the interactive output can be returned as a list. To return these datasets as a list, pass the "report" keyword on the `output` parameter. This list may in turn be passed to `proc_print` to write the report to a file.

Data Frame Output

The `proc_freq` function returns output datasets. If you are requesting only one table, a single data frame will be returned. If you request multiple tables, a list of data frames will be returned.

By default, the list items are named according to the strings specified on the `tables` parameter. You may control the names of the returned results by using a named vector on the `tables` parameter.

The standard output datasets are optimized for data manipulation. Column names have been standardized, and additional variables may be present to help with data manipulation. For instance, the by variable will always be named "BY", and the frequency category will always be named "CAT". In addition, data values in the output datasets are not rounded or formatted to give you the most accurate statistical results.

Frequency Weight

Normally the `proc_freq` function counts each row in the input data equally. In some cases, however, each row in the data can represent multiple observations, and rows should not be treated equally. In these cases, use the `weight` parameter. The parameter accepts a variable/column name to use as the weighted value. If the `weight` parameter is used, the function will sum the weighted values instead of counting rows.

By Groups

You may request that frequencies be separated into by groups using the by parameter. The parameter accepts one or more variable names from the input dataset. When this parameter is assigned, the data will be subset by the "by" variable(s) before frequency counts are calculated. On the interactive report, the by groups will appear in separate tables. On the output dataset, the by groups will be identified by additional columns.

Options

The `options` parameter accepts a vector of options. Normally, these options must be quoted. But you may pass them unquoted using the `v()` function. For example, you can request the number of category levels and the Chi-Square statistic like this: `options = v(nlevels, chisq)`.

Below are all the available options and a description of each:

- **crosstab**: Two-way output tables are a list style by default. If you want a crosstab style, pass the "crosstab" option.
- **list**: Two-way interactive tables are a crosstab style by default. If you want a list style two-way table, pass the "list" option.
- **missing**: Normally, missing values are not counted and not shown on frequency tables. The "missing" option allows you to treat missing (NA) values as normal values, so that they are counted and shown on the frequency table. Missing levels will appear on the table as a single dot (".").
- **nlevels**: The "nlevels" option will display the number of unique values for each variable in the frequency table. These levels are generated as a separate table that appears on the report, and will also be output from the function as a separate dataset.
- **nocol**: Two-way cross tabulation tables include column percents by default. To turn them off, pass the "nocol" option.

- **nocum**: Whether to include the cumulative frequency and percent columns on one-way, interactive tables. These columns are included by default. To turn them off, pass the "nocum" option.
- **nofreq**: The "nofreq" option will remove the frequency column from one-way and two-way tables.
- **nopercent**: The "nopercent" option will remove the percent column from one-way and two-way tables.
- **noprint**: Whether to print the interactive report to the viewer. By default, the report is printed to the viewer. The "noprint" option will inhibit printing.
- **nonobs**: Whether to include the number of observations "N" column on the output and interactive tables. By default, the N column will be included. The "nonobs" option turns it off.
- **norow**: Whether to include the row percentages on two-way crosstab tables. The "norow" option will turn them off.
- **nosparse/sparse**: Whether to include categories for which there are no frequency counts. Zero-count categories will be included by default, which is the "sparse" option. If the "nosparse" option is present, zero-count categories will be removed.
- **notable**: Whether to include the frequency table in the output dataset list. Normally, the frequency table is included. You may want to exclude the frequency table in some cases, for instance, if you only want the Chi-Square statistic.
- **outcum**: Whether to include the cumulative frequency and percent on output frequency tables. By default, these columns are not included. The "outcum" option will include them.

Statistics Options

In addition to the above options, the `options` parameter accepts some statistics options. The following keywords will generate an additional tables of specialized statistics. These statistics options are only available on two-way tables:

- **chisq**: Requests that the Chi-square statistics be produced.
- **fisher**: Requests that the Fisher's exact statistics be produced.

Using Factors

There are some occasions when you may want to define the `tables` variable or `by` variables as a factor. One occasion is for sorting/ordering, and the other is for obtaining zero-counts on sparse data.

To order the frequency categories in the frequency output, define the `tables` variable as a factor in the desired order. The function will then retain that order for the frequency categories in the output dataset and report.

You may also wish to define the tables variable as a factor if you are dealing with sparse data and some of the frequency categories are not present in the data. To ensure these categories are displayed with zero-counts, define the `tables` variable or `by` variable as a factor and use the "sparse" option. Note that the "sparse" option is actually the default.

If you do not want to show the zero-count categories on a variable that is defined as a factor, pass the "nosparse" keyword on the `options` parameter.

Ordering Frequency Output

Frequency output can be ordered using the `order` parameter. There are several choices:

- **internal**: The internal or "natural" order of the table variables. For character variables, "internal" means alphabetical order. For factors, "internal" means by order of the levels.
- **data**: "data" order is different from "internal" order. "data" order means the order that the data appears in the incoming data frame, which may or may not be the same as the "internal" order. This choice essentially sorts the frequencies according to however the table variable values were encountered in the incoming data.
- **freq**: The "freq" order value means to sort the output in descending frequency order.
- **formatted**: If there is a format assigned to the table variable(s), you may also use "formatted" order. "formatted" order means to apply the format to the table variable(s) and then sort the frequencies according to the order specified in the format. The format should be a user-defined format as defined in the **fmtr** package.

Note that the `order` parameter applies to both the interactive report and the data frame output. This behavior is different from SAS, which only orders the interactive report. The behavior of the `order` parameter was modified to reduce subsequent sorts and make it more convenient for the user.

Data Shaping

By default, the `proc_freq` function returns an output dataset of frequency results. If running interactively, the function also prints the frequency results to the viewer. As described above, the output dataset can be somewhat different than the dataset sent to the viewer. The `output` parameter allows you to choose which datasets to return, and how those datasets are structured.

There are three choices of datasets: "out", "report", and "none". The "out" keyword returns the default output dataset. The "report" keyword returns the dataset(s) sent to the viewer. You may also pass "none" if you don't want any datasets returned from the function.

In addition, the output dataset produced by the "out" keyword can be shaped in different ways. These shaping options allow you to decide whether the data should be returned long and skinny, or short and wide. The shaping options can reduce the amount of data manipulation necessary to get the frequencies into the desired form. The shaping options are as follows:

- **long**: Transposes the output datasets so that statistics are in rows and frequency categories are in columns.
- **stacked**: Requests that output datasets be returned in "stacked" form, such that both statistics and frequency categories are in rows.
- **wide**: Requests that output datasets be returned in "wide" form, such that statistics are across the top in columns, and frequency categories are in rows. This shaping option is the default.

Plots

The `proc_freq` function can generate plots. These plots can be most easily generated by setting `plots = TRUE`. The default plots are bar charts showing the frequency counts for each category and each table request. For two-way interactions, the function will plot a bar chart for each interaction.

In addition to standard bar charts, the function can produce dot plots, stacked bar charts, and clustered bar charts. To request these types of plots, pass a `freqplot` function to the `plots` parameter.

The freqplot function provides several optional parameters for customizing your chart. See the freqplot documentation for a full list of options and more plot examples.

See Also

For summary statistics, see proc_means. To pivot or transpose the data coming from proc_freq, see proc_transpose. For frequency plots, see freqplot.

Examples

```
library(procs)

# Turn off printing for CRAN checks
options("procs.print" = FALSE)

# Create sample data
df <- as.data.frame(HairEyeColor, stringsAsFactors = FALSE)

# Assign labels
labels(df) <- list(Hair = "Hair Color",
                   Eye = "Eye Color",
                   Sex = "Sex at Birth")

# Example #1: One way frequencies on Hair and Eye color with weight option.
res <- proc_freq(df,
                 tables = v(Hair, Eye),
                 options = outcum,
                 weight = Freq)

# View result data
res
# $Hair
#    VAR   CAT   N CNT      PCT CUMSUM    CUMPCT
# 1 Hair Black 592 108 18.24324    108  18.24324
# 2 Hair Blond 592 127 21.45270    235  39.69595
# 3 Hair Brown 592 286 48.31081    521  88.00676
# 4 Hair   Red 592  71 11.99324    592 100.00000
#
# $Eye
#   VAR   CAT   N CNT      PCT CUMSUM    CUMPCT
# 1 Eye  Blue 592 215 36.31757    215  36.31757
# 2 Eye Brown 592 220 37.16216    435  73.47973
# 3 Eye Green 592  64 10.81081    499  84.29054
# 4 Eye Hazel 592  93 15.70946    592 100.00000

# Example #2: 2 x 2 Crosstabulation table with Chi-Square statistic
res <- proc_freq(df, tables = Hair * Eye,
                     weight = Freq,
                     options = v(crosstab, chisq))

# View result data
res
```

```
#$`Hair * Eye`
#   Category Statistic       Blue      Brown      Green     Hazel     Total
#1     Black Frequency  20.000000  68.000000  5.0000000 15.000000 108.00000
#2     Black   Percent   3.378378  11.486486  0.8445946  2.533784  18.24324
#3     Black   Row Pct  18.518519  62.962963  4.6296296 13.888889        NA
#4     Black   Col Pct   9.302326  30.909091  7.8125000 16.129032        NA
#5     Blond Frequency  94.000000   7.000000 16.0000000 10.000000 127.00000
#6     Blond   Percent  15.878378   1.182432  2.7027027  1.689189  21.45270
#7     Blond   Row Pct  74.015748   5.511811 12.5984252  7.874016        NA
#8     Blond   Col Pct  43.720930   3.181818 25.0000000 10.752688        NA
#9     Brown Frequency  84.000000 119.000000 29.0000000 54.000000 286.00000
#10    Brown   Percent  14.189189  20.101351  4.8986486  9.121622  48.31081
#11    Brown   Row Pct  29.370629  41.608392 10.1398601 18.881119        NA
#12    Brown   Col Pct  39.069767  54.090909 45.3125000 58.064516        NA
#13      Red Frequency  17.000000  26.000000 14.0000000 14.000000  71.00000
#14      Red   Percent   2.871622   4.391892  2.3648649  2.364865  11.99324
#15      Red   Row Pct  23.943662  36.619718 19.7183099 19.718310        NA
#16      Red   Col Pct   7.906977  11.818182 21.8750000 15.053763        NA
#17    Total Frequency 215.000000 220.000000 64.0000000 93.000000 592.00000
#18    Total   Percent  36.317568  37.162162 10.8108108 15.709459 100.00000

# $`chisq:Hair * Eye`
#                          STAT DF      VAL         PROB
# 1                 Chi-Square  9 138.2898 2.325287e-25
# 2 Continuity Adj. Chi-Square  9 138.2898 2.325287e-25

# Example #3: By variable with named table request
res <- proc_freq(df, tables = v(Hair, Eye, Cross = Hair * Eye),
                 by = Sex,
                 weight = Freq)

# View result data
res
# $Hair
#       BY  VAR   CAT   N CNT      PCT
# 1 Female Hair Black 313  52 16.61342
# 2 Female Hair Blond 313  81 25.87859
# 3 Female Hair Brown 313 143 45.68690
# 4 Female Hair   Red 313  37 11.82109
# 5   Male Hair Black 279  56 20.07168
# 6   Male Hair Blond 279  46 16.48746
# 7   Male Hair Brown 279 143 51.25448
# 8   Male Hair   Red 279  34 12.18638
#
# $Eye
#       BY VAR   CAT   N CNT       PCT
# 1 Female Eye  Blue 313 114 36.421725
# 2 Female Eye Brown 313 122 38.977636
# 3 Female Eye Green 313  31  9.904153
# 4 Female Eye Hazel 313  46 14.696486
# 5   Male Eye  Blue 279 101 36.200717
# 6   Male Eye Brown 279  98 35.125448
# 7   Male Eye Green 279  33 11.827957
```

```
# 8   Male Eye Hazel 279  47 16.845878
#
# $Cross
#        BY VAR1 VAR2  CAT1  CAT2   N CNT        PCT
# 1  Female Hair  Eye Black  Blue 313   9  2.8753994
# 2  Female Hair  Eye Black Brown 313  36 11.5015974
# 3  Female Hair  Eye Black Green 313   2  0.6389776
# 4  Female Hair  Eye Black Hazel 313   5  1.5974441
# 5  Female Hair  Eye Blond  Blue 313  64 20.4472843
# 6  Female Hair  Eye Blond Brown 313   4  1.2779553
# 7  Female Hair  Eye Blond Green 313   8  2.5559105
# 8  Female Hair  Eye Blond Hazel 313   5  1.5974441
# 9  Female Hair  Eye Brown  Blue 313  34 10.8626198
# 10 Female Hair  Eye Brown Brown 313  66 21.0862620
# 11 Female Hair  Eye Brown Green 313  14  4.4728435
# 12 Female Hair  Eye Brown Hazel 313  29  9.2651757
# 13 Female Hair  Eye   Red  Blue 313   7  2.2364217
# 14 Female Hair  Eye   Red Brown 313  16  5.1118211
# 15 Female Hair  Eye   Red Green 313   7  2.2364217
# 16 Female Hair  Eye   Red Hazel 313   7  2.2364217
# 17   Male Hair  Eye Black  Blue 279  11  3.9426523
# 18   Male Hair  Eye Black Brown 279  32 11.4695341
# 19   Male Hair  Eye Black Green 279   3  1.0752688
# 20   Male Hair  Eye Black Hazel 279  10  3.5842294
# 21   Male Hair  Eye Blond  Blue 279  30 10.7526882
# 22   Male Hair  Eye Blond Brown 279   3  1.0752688
# 23   Male Hair  Eye Blond Green 279   8  2.8673835
# 24   Male Hair  Eye Blond Hazel 279   5  1.7921147
# 25   Male Hair  Eye Brown  Blue 279  50 17.9211470
# 26   Male Hair  Eye Brown Brown 279  53 18.9964158
# 27   Male Hair  Eye Brown Green 279  15  5.3763441
# 28   Male Hair  Eye Brown Hazel 279  25  8.9605735
# 29   Male Hair  Eye   Red  Blue 279  10  3.5842294
# 30   Male Hair  Eye   Red Brown 279  10  3.5842294
# 31   Male Hair  Eye   Red Green 279   7  2.5089606
# 32   Male Hair  Eye   Red Hazel 279   7  2.5089606

# Example #4: Default Plots
res <- proc_freq(df, tables = v(Hair, Eye, Cross = Hair * Eye),
                 weight = Freq,
                 plots = TRUE,    # Request default plots
                 output = report  # So we can see plot objects
                 )

# View results
# Frequency plots are created for each table
res
# $Hair
# CAT   N CNT      PCT CUMSUM    CUMPCT
# 1 Black 592 108 18.24324    108  18.24324
# 2 Blond 592 127 21.45270    235  39.69595
# 3 Brown 592 286 48.31081    521  88.00676
# 4   Red 592  71 11.99324    592 100.00000
```

```
#
# $`Plots: Hair`
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa8212f57c7.jpg'
# - height: 4.5
# - width: 6
#
# $Eye
# CAT   N CNT      PCT CUMSUM    CUMPCT
# 1  Blue 592 215 36.31757    215  36.31757
# 2 Brown 592 220 37.16216    435  73.47973
# 3 Green 592  64 10.81081    499  84.29054
# 4 Hazel 592  93 15.70946    592 100.00000
#
# $`Plots: Eye`
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa854b429fc.jpg'
# - height: 4.5
# - width: 6
#
# $Cross
# CAT Statistic       Blue      Brown      Green     Hazel     Total
# 1  Black Frequency  20.000000  68.000000  5.0000000 15.000000 108.00000
# 2  Black   Percent   3.378378  11.486486  0.8445946  2.533784  18.24324
# 3  Black   Row Pct  18.518519  62.962963  4.6296296 13.888889        NA
# 4  Black   Col Pct   9.302326  30.909091  7.8125000 16.129032        NA
# 5  Blond Frequency  94.000000   7.000000 16.0000000 10.000000 127.00000
# 6  Blond   Percent  15.878378   1.182432  2.7027027  1.689189  21.45270
# 7  Blond   Row Pct  74.015748   5.511811 12.5984252  7.874016        NA
# 8  Blond   Col Pct  43.720930   3.181818 25.0000000 10.752688        NA
# 9  Brown Frequency  84.000000 119.000000 29.0000000 54.000000 286.00000
# 10 Brown   Percent  14.189189  20.101351  4.8986486  9.121622  48.31081
# 11 Brown   Row Pct  29.370629  41.608392 10.1398601 18.881119        NA
# 12 Brown   Col Pct  39.069767  54.090909 45.3125000 58.064516        NA
# 13   Red Frequency  17.000000  26.000000 14.0000000 14.000000  71.00000
# 14   Red   Percent   2.871622   4.391892  2.3648649  2.364865  11.99324
# 15   Red   Row Pct  23.943662  36.619718 19.7183099 19.718310        NA
# 16   Red   Col Pct   7.906977  11.818182 21.8750000 15.053763        NA
# 17 Total Frequency 215.000000 220.000000 64.0000000 93.000000 592.00000
# 18 Total   Percent  36.317568  37.162162 10.8108108 15.709459 100.00000
#
# $`Plots: Hair * Eye`
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa87b6a3303.jpg'
# - height: 4.5
# - width: 6
```

---

proc_means *Calculates Summary Statistics*

Description

The `proc_means` function generates summary statistics for selected variables on the input dataset. The variables are identified on the `var` parameter. The statistics to perform are identified on the `stats` parameter. Results are displayed in the viewer interactively and returned from the function.

Usage

```
proc_means(
  data,
  var = NULL,
  stats = c("n", "mean", "std", "min", "max"),
  output = NULL,
  by = NULL,
  class = NULL,
  weight = NULL,
  options = NULL,
  titles = NULL,
  where = NULL
)
```

Arguments

`data`	The input data frame for which to calculate summary statistics. This parameter is required.
`var`	The variable(s) to calculate summary statistics for. If no variables are specified, summary statistics will be generated for all numeric variables on the input data frame.
`stats`	A vector of summary statistics keywords. Valid keywords are: "css", "clm", "cv", "kurt", "kurtosis", "lclm", "mean", "median", "mode", "min", "max", "n", "nmiss", "nobs", "p1", "p5", "p10", "p20", "p25", "p30", "p40", "p50", "p60", "p70", "p75", "p80", "p90", "p95", "p99", "q1", "q3", "qrange", "range", "skew", "skewness", "std", "stddev", "stderr", "sum", "uclm", "uss", and "vari". For hypothesis testing, the function supports "t", "prt", "probt", and "df". Default statistics are: "n", "mean", "std", "min", and "max".
`output`	Whether or not to return datasets from the function. Valid values are "out", "none", and "report". Default is "out", and will produce dataset output specifically designed for programmatic use. The "none" option will return a NULL instead of a dataset or list of datasets. The "report" keyword returns the datasets from the interactive report, which may be different from the standard output. The output parameter also accepts data shaping keywords "long, "stacked", and "wide". The shaping keywords control the structure of the output data. See the **Data Shaping** section for additional details. Note that multiple output keywords may be passed on a character vector. For example, to produce both a report dataset and a "long" output dataset, use the parameter `output = c("report", "out", "long")`.
`by`	An optional by group. If you specify a by group, the input data will be subset on the by variable(s) prior to performing any statistics.

`class` The `class` parameter is similar to the `by` parameter, but the output is different. By groups will create completely separate tables, while class groups will be continued in the same table. When a `by` and a `class` are both specified, the `class` will be nested in the `by`.

`weight` The name of a variable to use for weighted statistics. This parameter is optional, and NULL by default. The weight can be used to calculate weighted means, sums, variances, etc. Note that any observations with an NA value for the weight will be excluded from the analysis. Also observe the "vardef" option for weighted variances, as this option can effect which denominator is used for the variance calculation.

`options` A vector of optional keywords. Valid values are: "alpha =", "completetypes", "maxdec =", "noprint", "notype", "nofreq", "nonobs", "nway", "vardef=". The "notype", "nofreq" and "nonobs" keywords will turn off columns on the output datasets. The "alpha = " option will set the alpha value for confidence limit statistics. The default is 95% (alpha = 0.05). The "maxdec = " option sets the maximum number of decimal places displayed on report output. The "nway" option returns only the highest type values. The "vardef=" option specifies the variance divisor. Note that the selected variance divisor can effect the calculations for several statistics. Valid values for "vardef=" are: "DF", "N", "WEIGHT", "WGT", or "WDF". See details for further explanation of these values.

`titles` A vector of one or more titles to use for the report output.

`where` An expression to filter the rows before the statistics are calculated. Use the `expression` function to define the filter.

Details

The `proc_means` function is for analysis of continuous variables. Data is passed in on the `data` parameter. The desired statistics are specified using keywords on the `stats` parameter. The function can segregate data into groups using the `by` and `class` parameters. There are also options to determine whether and what results are returned.

Value

Normally, the requested summary statistics are shown interactively in the viewer, and output results are returned as a data frame. If the request produces multiple data frames, they will be returned in a list. You may then access individual datasets from the list. The interactive report can be turned off using the "noprint" option, and the output datasets can be turned off using the "none" keyword on the `output` parameter.

Interactive Output

By default, `proc_freq` results will be immediately sent to the viewer as an HTML report. This functionality makes it easy to get a quick analysis of your data. To turn off the interactive report, pass the "noprint" keyword to the `options` parameter.

The `titles` parameter allows you to set one or more titles for your report. Pass these titles as a vector of strings.

The exact datasets used for the interactive report can be returned as a list. To return these datasets as a list, pass the "report" keyword on the `output` parameter. This list may in turn be passed to `proc_print` to write the report to a file.

Dataset Output

Dataset results are also returned from the function by default. If the results are a single dataset, a single data frame will be returned. If there are multiple results, a list of data frames will be returned.

The output datasets generated are optimized for data manipulation. The column names have been standardized, and additional variables may be present to help with data manipulation. For example, the by variable will always be named "BY", and the class variable will always be named "CLASS". In addition, data values in the output datasets are intentionally not rounded or formatted to give you the most accurate statistical results.

Statistics Keywords

The following statistics keywords can be passed on the `stats` parameter. Normally, each statistic will be contained in a separate column and the column name will be the same as the statistic keyword. You may pass statistic keywords as a quoted vector of strings, or an unquoted vector using the `v()` function.

- **css**: Corrected Sum of Squares.
- **clm, lclm, uclm**: Upper and lower confidence limits.
- **cv**: Coefficient of Variation.
- **kurt/kurtosis**: The Kurtosis is a description of the distribution tails. It requires at least 4 complete observations.
- **mean**: The arithmetic mean.
- **median**: The median.
- **mode**: The mode of the target variable.
- **min, max**: The minimum and maximum values of the target variable.
- **n**: The number of non-missing observations.
- **nmiss**: The number of missing observations.
- **nobs**: The number of observations, whether missing or non-missing.
- **p1 - p99**: Percentile ranges from p1 to p99, in increments of 5.
- **qrange, q1, q3**: Quantile ranges for the first and third quantiles.
- **range**: Difference between the minimum and maximum values.
- **skew/skewness**: A measure of distribution skewness. It requires at least 3 complete observations.
- **std/stddev**: Standard deviation.
- **stderr**: Standard error.
- **sum**: The sum of variable values.
- **uss**: Uncorrected sum of squares.
- **vari**: The variance.

The function supports the following keywords to perform hypothesis testing:

- **t**: Student's t statistic.
- **prt/probt**: A two-tailed p-value for the Student's t statistic.
- **df**: Degrees of freedom for the Student's t statistic.

Weight

The function can produce weighted statistics using the `weight` parameter. A weight assigns a relative importance to an observation. To produce weighted statistics, pass the name of the variable that holds the weight to the `weight` parameter, quoted or unquoted. This variable will then be used when calculating many of the statistics generated by `proc_means`. For example, the "mean", "median", "std", and "vari" statistics will accept weighted values. On the other hand, "min" and "max" are not effected by the weight. The "skewness" and "kurtosis" statistics also do not not accept weighted values, and will instead produce an NA.

Options

The `proc_means` function recognizes the following options. Options may be passed as a quoted vector of strings, or an unquoted vector using the `v()` function.

- **alpha =** : The "alpha = " option will set the alpha value for confidence limit statistics. Set the alpha as a decimal value between 0 and 1. For example, you can set a 90% confidence limit as `alpha = 0.1`.
- **completetypes**: The "completetypes" option will generate all combinations of the class variable, even if there is no data present for a particular level. Combinations will be distinguished by the TYPE variable. To use the "completetypes" option, define the class variable(s) as a factor.
- **maxdec =** : The "maxdec = " option will set the maximum of decimal places displayed on report output. For example, you can set 4 decimal places as follows: `maxdec = 4`. Default is 7 decimal places. This option will not round any values on the output dataset.
- **nofreq, nonobs**: Turns off the FREQ column on the output datasets.
- **noprint**: Whether to print the interactive report to the viewer. By default, the report is printed to the viewer. The "noprint" option will inhibit printing.
- **notype**: Turns off the TYPE column on the output dataset.
- **nway**: Returns only the highest level TYPE combination. By default, the function returns all TYPE combinations.
- **vardef**: Controls the denominator used in variance-related statistics. This option supports the following denominator definitions:
  - **DF**: Uses \(n - 1\), the sample degrees of freedom.
  - **N**: Uses \(n\), the total count of observations.
  - **WEIGHT/WGT**: Uses the sum of weights.
  - **WDF**: Uses the sum of weights minus one.

  By default, this option use **DF**. The variance divisor effects many statistics, as many statistics use the variance in their calculations. Also note that some divisors are not supported by some statistics. For those statistics that are not supported by a particular divisor, they will be returned as NA.

TYPE and FREQ Variables

The TYPE and FREQ variables appear on the output dataset by default.

The FREQ variable contains a count of the number of input rows/observations that were included in the statistics for that output row. The FREQ count can be different from the N statistic. The FREQ count is a count of the number of rows/observations, while the N statistic is a count of non-missing values. These counts can be different if you have missing values in your data. If you want to remove the FREQ column from the output dataset, use the "nofreq" option.

The TYPE variable identifies combinations of class categories, and produces summary statistics for each of those combinations. For example, the output dataset normally produces statistics for TYPE 0, which is all class categories, and a TYPE 1 which is each class category. If there are multiple classes, there will be multiple TYPE values for each level of class combinations. If you do no want to show the various type combinations, use the "nway" option. If you want to remove the TYPE column from the output dataset, use the "notype" option.

Using Factors

There are some occasions when you may want to define the `class` variable(s) as a factor. One occasion is for sorting/ordering, and the other is for obtaining zero-counts on sparse data.

To order the class categories in the means output, define the `class` variable as a factor in the desired order. The function will then retain that order for the class categories in the output dataset and report.

You may also wish to define the class variable as a factor if you are dealing with sparse data and some of the class categories are not present in the data. To ensure these categories are displayed with zero-counts, define the `class` variable as a factor and use the "completetypes" option.

Data Shaping

The output dataset produced by the "out" keyword can be shaped in different ways. These shaping options allow you to decide whether the data should be returned long and skinny, or short and wide. The shaping options can reduce the amount of data manipulation necessary to get the frequencies into the desired form. The shaping options are as follows:

- **long**: Transposes the output datasets so that statistics are in rows and variables are in columns.
- **stacked**: Requests that output datasets be returned in "stacked" form, such that both statistics and variables are in rows.
- **wide**: Requests that output datasets be returned in "wide" form, such that statistics are across the top in columns, and variables are in rows. This shaping option is the default.

Examples

```
# Turn off printing for CRAN checks
options("procs.print" = FALSE)

# Default statistics on iris
res1 <- proc_means(iris)

# View results
res1
```

```
#   TYPE FREQ          VAR   N     MEAN       STD MIN MAX
# 1    0  150 Sepal.Length 150 5.843333 0.8280661 4.3 7.9
# 2    0  150  Sepal.Width 150 3.057333 0.4358663 2.0 4.4
# 3    0  150 Petal.Length 150 3.758000 1.7652982 1.0 6.9
# 4    0  150  Petal.Width 150 1.199333 0.7622377 0.1 2.5

# Defaults statistics with by
res2 <- proc_means(iris,
                   by = Species)
# View results
res2
#            BY TYPE FREQ          VAR  N  MEAN       STD MIN MAX
# 1      setosa    0   50 Sepal.Length 50 5.006 0.3524897 4.3 5.8
# 2      setosa    0   50  Sepal.Width 50 3.428 0.3790644 2.3 4.4
# 3      setosa    0   50 Petal.Length 50 1.462 0.1736640 1.0 1.9
# 4      setosa    0   50  Petal.Width 50 0.246 0.1053856 0.1 0.6
# 5  versicolor    0   50 Sepal.Length 50 5.936 0.5161711 4.9 7.0
# 6  versicolor    0   50  Sepal.Width 50 2.770 0.3137983 2.0 3.4
# 7  versicolor    0   50 Petal.Length 50 4.260 0.4699110 3.0 5.1
# 8  versicolor    0   50  Petal.Width 50 1.326 0.1977527 1.0 1.8
# 9   virginica    0   50 Sepal.Length 50 6.588 0.6358796 4.9 7.9
# 10  virginica    0   50  Sepal.Width 50 2.974 0.3224966 2.2 3.8
# 11  virginica    0   50 Petal.Length 50 5.552 0.5518947 4.5 6.9
# 12  virginica    0   50  Petal.Width 50 2.026 0.2746501 1.4 2.5

# Specified variables, statistics, and options
res3 <- proc_means(iris,
                   var = v(Petal.Length, Petal.Width),
                   class = Species,
                   stats = v(n, mean, std, median, qrange, clm),
                   options = nofreq,
                   output = long)
# View results
res3
#         CLASS TYPE   STAT Petal.Length Petal.Width
# 1        <NA>    0      N  150.0000000 150.0000000
# 2        <NA>    0   MEAN    3.7580000   1.1993333
# 3        <NA>    0    STD    1.7652982   0.7622377
# 4        <NA>    0 MEDIAN    4.3500000   1.3000000
# 5        <NA>    0 QRANGE    3.5000000   1.5000000
# 6        <NA>    0   LCLM    3.4731854   1.0763533
# 7        <NA>    0   UCLM    4.0428146   1.3223134
# 8      setosa    1      N   50.0000000  50.0000000
# 9      setosa    1   MEAN    1.4620000   0.2460000
# 10     setosa    1    STD    0.1736640   0.1053856
# 11     setosa    1 MEDIAN    1.5000000   0.2000000
# 12     setosa    1 QRANGE    0.2000000   0.1000000
# 13     setosa    1   LCLM    1.4126452   0.2160497
# 14     setosa    1   UCLM    1.5113548   0.2759503
# 15 versicolor    1      N   50.0000000  50.0000000
# 16 versicolor    1   MEAN    4.2600000   1.3260000
# 17 versicolor    1    STD    0.4699110   0.1977527
# 18 versicolor    1 MEDIAN    4.3500000   1.3000000
```

```
# 19 versicolor    1 QRANGE    0.6000000   0.3000000
# 20 versicolor    1   LCLM    4.1264528   1.2697993
# 21 versicolor    1   UCLM    4.3935472   1.3822007
# 22  virginica    1       N   50.0000000  50.0000000
# 23  virginica    1   MEAN    5.5520000   2.0260000
# 24  virginica    1    STD    0.5518947   0.2746501
# 25  virginica    1 MEDIAN    5.5500000   2.0000000
# 26  virginica    1 QRANGE    0.8000000   0.5000000
# 27  virginica    1   LCLM    5.3951533   1.9479453
# 28  virginica    1   UCLM    5.7088467   2.1040547
```

`proc_print` *Prints a dataset*

Description

The `proc_print` function is used to print a dataset or datasets. To print multiple datasets, pass them to the `data` parameter in a list. By default, the function prints to the viewer. It may also be used to print to the file system using the `output_type` and `file_path` parameters. This print function has limited options, and is meant to quickly view your data or dump it out to a file. For more reporting options, use the **reporter** package.

Usage

```
proc_print(
  data,
  file_path = NULL,
  output_type = "HTML",
  titles = NULL,
  style = NULL,
  view = TRUE
)
```

Arguments

`data`	The data to print. Can be either a single dataset, or a list of datasets.
`file_path`	The path of the report to print.
`output_type`	The type of report to create. Valid values are "TXT", "RTF", "PDF", "HTML", and "DOCX". Default is "HTML".
`titles`	A vector of titles.
`style`	A style object, as defined by the **reporter** package. See create_style for details.
`view`	Whether to send the print output to the viewer. Valid values are TRUE and FALSE. Default is TRUE.

Value

If a file report was produced, the full path of the report. Otherwise, a NULL. In either case, the value will be returned invisibly.

Examples

```
# Turn off printing to pass CRAN checks
options("procs.print" = FALSE)

# Print mtcars to the viewer
proc_print(mtcars)

# Print mtcars to an RTF
pth <- proc_print(mtcars,
                  file_path = tempfile(fileext = ".rtf"),
                  titles = "MTCARS Proc Print Example",
                  output_type = "RTF", view = FALSE)

# View file
# file.show(pth)
```

---

proc_reg *Calculates a Regression*

Description

The `proc_reg` function performs a regression for one or more models. The model(s) are passed on the `model` parameter, and the input dataset is passed on the `data` parameter. The `stats` parameter allows you to request additional statistics, similar to the model options in SAS. The `by` parameter allows you to subset the data into groups and run the model on each group. The `weight` parameter lets you assign a weight to each observation in the dataset. The `output` and `options` parameters provide additional customization of the results.

Usage

```
proc_reg(
  data,
  model,
  by = NULL,
  stats = NULL,
  output = NULL,
  weight = NULL,
  options = NULL,
  titles = NULL,
  plots = NULL,
  where = NULL
)
```

Arguments

`data`	The input data frame for which to perform the regression analysis. This parameter is required.
`model`	A model for the regression to be performed. The model can be specified using either R syntax or SAS syntax. `model = var1 ~ var2 + var3` is an example of R style model syntax. If you wish to pass multiple models using R syntax, pass them in a list. For SAS syntax, pass the model as a quoted string: `model = "var1 = var2 var3"`. To pass multiple models using SAS syntax, pass them as a vector of strings. By default, the models will be named "MODEL1", "MODEL2", etc. If you want to name your model, pass it as a named list or named vector.
`by`	An optional by group. If you specify a by group, the input data will be subset on the by variable(s) prior to performing the regression. For multiple by variables, pass them as a quoted vector of variable names. You may also pass them unquoted using the `v` function.
`stats`	Optional statistics keywords. Valid values are "adjrsq", "aic", "clb", "est", "edf", "hcc", "hccmethod", "mse", "p", "press", "rsquare", "sse", "spec", "seb", and "table". A single keyword may be passed with or without quotes. Pass multiple keywords either as a quoted vector, or unquoted vector using the `v()` function. These statistics keywords largely correspond to the options on the "model" statement in SAS. Most of them control which statistics are added to the interactive report. Some keywords control statistics on the output dataset. See the **Statistics Keywords** section for details on the purpose and target of each keyword.
`output`	Whether or not to return datasets from the function. Valid values are "out", "none", and "report". Default is "out", and will produce dataset output specifically designed for programmatic use. The "none" option will return a NULL instead of a dataset or list of datasets. The "report" keyword returns the datasets from the interactive report, which may be different from the standard output. Note that some statistics are only available on the interactive report. The output parameter also accepts data shaping keywords "long, "stacked", and "wide". These shaping keywords control the structure of the output data. See the **Data Shaping** section for additional details. Note that multiple output keywords may be passed on a character vector. For example, to produce both a report dataset and a "long" output dataset, use the parameter `output = c("report", "out", "long")`.
`weight`	The name of a variable to use as a weight for each observation. The weight is commonly provided as the inverse of each variance.
`options`	A vector of optional keywords. Valid values are: "alpha =", "edf", "noprint", "outest", "outseb", "press", "rsquare", and "tableout". The "alpha = " option will set the alpha value for confidence limit statistics. The default is 95% (alpha = 0.05). The "noprint" option turns off the interactive report. For other options, see the **Options** section for explanations of each.
`titles`	A vector of one or more titles to use for the report output.
`plots`	Pass the desired plot(s) on this parameter. Valid values are TRUE, "all", a vector of plot names, or a call to the `regplot` function. Default is NULL, meaning no plots are desired. If there are multiple model requests, you can pass a single

plot request which will apply to all models, or a list of plot requests that aligns one-to-one for each model formula.

`where` An expression to filter the rows before statistics are calculated. Use the `expression` function to define the filter.

Details

The `proc_reg` function is a general-purpose regression function. It produces a dataset output by default, and, when working in RStudio, also produces an interactive report. The function has many convenient options for what statistics are produced and how the analysis is performed. All statistical output from `proc_reg` matches SAS.

A model may be specified using R model syntax or SAS model syntax. To use SAS syntax, the model statement must be quoted. To pass multiple models using R syntax, pass them to the `model` parameter in a list. To pass multiple models using SAS syntax, pass them to the `model` parameter as a vector of strings.

Value

Normally, the requested regression statistics are shown interactively in the viewer, and output results are returned as a data frame. If you request "report" datasets, they will be returned as a list. You may then access individual datasets from the list using dollar sign ($) syntax. The interactive report can be turned off using the "noprint" option. The output dataset can be turned off using the "none" keyword on the `output` parameter. If the output dataset is turned off, the function will return a NULL.

Interactive Output

By default, `proc_reg` results will be sent to the viewer as an HTML report. This functionality makes it easy to get a quick analysis of your data. To turn off the interactive report, pass the "noprint" keyword to the `options` parameter.

The `titles` parameter allows you to set one or more titles for your report. Pass these titles as a vector of strings.

The exact datasets used for the interactive report can be returned as a list. To return these datasets, pass the "report" keyword on the `output` parameter. This list may in turn be passed to `proc_print` to write the report to a file.

Dataset Output

Dataset results are also returned from the function by default. `proc_reg` typically returns a single dataset. The columns and rows on this dataset can change depending on the keywords passed to the `stats` and `options` parameters.

The default output dataset is optimized for data manipulation. The column names have been standardized, and additional variables may be present to help with data manipulation. The data values in the output dataset are intentionally not rounded or formatted to give you the most accurate numeric results.

You may also request to return the datasets used in the interactive report. To request these datasets, pass the "report" option to the `output` parameter. Each report dataset will be named according to

the category of statistical results. There are four standard categories: "NObs", "ANOVA", "FitStatistics", and "ParameterEstimates". When the "spec" statistics option is passed, the function will also return a "SpecTest" dataset containing the White's test results. If the "p" option is present, the "OutputStatistics" and "ResidualStatistics" tables will be included.

If you don't want any datasets returned, pass the "none" option on the `output` parameter.

Statistics Keywords

The following statistics keywords can be passed on the `stats` parameter. You may pass statistic keywords as a quoted vector of strings, or an unquoted vector using the `v()` function. An individual statistics keyword can be passed without quoting.

- **adjrsq**: Adds adjusted r-square value to the output dataset.
- **clb**: Requests confidence limits be added to the interactive report.
- **edf**: Includes the number of regressors, the error degress of freedom, and the model r-square to the output dataset.
- **est**: Request an output dataset of parameter estimate and optional model fit summary statistics. This statistics option is the default.
- **hcc**: The "hcc" statistics keyword requests that heteroscedasticity-consistent standard errors of the parameter estimates be sent to the interactive report.
- **hccmethod=**: When the "hcc" option is present, the "hccmethod=" option specifies the type of method to use. Valid values are 0 and 3.
- **mse**: Computes the mean squared error for each model and adds to the output dataset.
- **p**: Computes predicted and residual values and sends to a separate table on the interactive report.
- **press**: Includes the predicted residual sum of squares (PRESS) statistic in the output dataset.
- **rsquare**: Include the r-square statistic in the output dataset. The "rsquare" option has the same effect as the "edf" option.
- **seb**: Outputs the standard errors of the parameter estimates to the output dataset. These values will be identified as type "SEB".
- **spec**: Adds the "White's test" table to the interactive output. This test determines whether the first and second moments of the model are correctly specified.
- **sse**: Adds the error sum of squares to the output dataset.
- **table**: The "table" keyword is used to send standard errors, t-statistics, p-values, and confidence limits to the output dataset. These additional statistics are identified by types "STDERR", "T", and "PVALUE". The confidence limits are identified by "LxxB" and "UxxB", where "xx" is the alpha value in percentage terms. The "table" keyword on the `stats` parameter performs the same functions as the "tableout" option on the `options` parameter.

Options

The `proc_reg` function recognizes the following options. Options may be passed as a quoted vector of strings, or an unquoted vector using the `v()` function.

- **alpha =** : The "alpha = " option will set the alpha value for confidence limit statistics. Set the alpha as a decimal value between 0 and 1. For example, you can set a 90% confidence limit as `alpha = 0.1`.
- **edf**: Includes the number of regressors, the error degress of freedom, and the model r-square to the output dataset.
- **noprint**: Whether to print the interactive report to the viewer. By default, the report is printed to the viewer. The "noprint" option will inhibit printing. You may inhibit printing globally by setting the package print option to false: `options("procs.print" = FALSE)`.
- **outest**: The "outest" option is used to request the parameter estimates and model fit summary statistic be sent to the output dataset. The parameter estimates are identified as type "PARMS" on the output dataset. The "outest" dataset is the default output dataset, and this option does not normally need to be passed.
- **outseb**: The "outseb" option is used to request the standard errors be sent to the output dataset. The standard errors will be added as a new row identified by type "SEB". This request can also be made by passing the "seb" keyword to the `stats` parameter.
- **press**: Includes the predicted residual sum of squares (PRESS) statistic in the output dataset.
- **rsquare**: Include the r-square statistic in the output dataset. The "rsquare" option has the same effect as the "edf" option.
- **tableout**: The "tableout" option is used to send standard errors, t-statistics, p-values, and confidence limits to the output dataset. These additional statistics are identified by types "STDERR", "T", and "PVALUE". The confidence limits are identified by "LxxB" and "UxxB", where "xx" is the alpha value in percentage terms. The "tableout" option on the `options` parameter performs the same functions as the "table" keyword on the `stats` parameter.

Data Shaping

The output datasets produced by the function can be shaped in different ways. These shaping options allow you to decide whether the data should be returned long and skinny, or short and wide. The shaping options can reduce the amount of data manipulation necessary to get the data into the desired form. The shaping options are as follows:

- **long**: Transposes the output datasets so that statistics are in rows and variables are in columns.
- **stacked**: Requests that output datasets be returned in "stacked" form, such that both statistics and variables are in rows.
- **wide**: Requests that output datasets be returned in "wide" form, such that statistics are across the top in columns, and variables are in rows. This shaping option is the default.

These shaping options are passed on the `output` parameter. For example, to return the data in "long" form, use `output = "long"`.

Plots

The `plots` parameter allows you to request several types of regression plot. Below are the types of plots that are supported. The list shows the plot type keyword needed to request the plot, and a brief description:

- **diagnostics**: A fit diagnostics panel that contains 8 different types of plots and a table of statistics.

- **cooksd**: Cook's D statistic vs. Observation number.
- **dfbetas**: Displays the influence of each observation for each coefficient in the model.
- **dffits**: Displays the influence of each observation on fitted values.
- **fitplot**: Produces a scatter plot of the dependent variable against the regressor, including the fitted line and confidence/prediction bands. This is only available for models with a single regressor.
- **observedbypredicted**: Dependent variable (Observed) vs. Predicted values.
- **qqplot**: Normal Quantile-Quantile (Q-Q) plot of residuals.
- **residuals**: Produces a panel of residual plots against each independent variable in the model.
- **residualbypredicted**: Residuals vs. Predicted values.
- **residualhistogram**: Histogram of residuals, with a normal and kernel curve overlay.
- **rfplot**: Residual-Fit (RF) spread plot.
- **rstudentbyleverage**: Externally Studentized Residuals vs. Leverage.
- **rstudentbypredicted**: Externally Studentized Residuals (RStudent) vs. Predicted values.

The above plots may be requested in different ways: as a vector of keywords, or as a call to the regplot function. The keyword approach will produce plots with a default configuration for each type of plot . A call to regplot will give you control over some plot options. See the regplot function for further details.

See Also

regplot()

Examples

```
# Turn off printing for CRAN checks
options("procs.print" = FALSE)

# Prepare sample data
set.seed(123)
dat <- cars
samplecar <- sample(c(TRUE, FALSE), nrow(cars), replace=TRUE, prob=c(0.6, 0.4))
dat$group <- ifelse(samplecar %in% seq(1, nrow(cars)), "Group A", "Group B")

# Example 1: R Model Syntax
res1 <- proc_reg(dat, model = dist ~ speed)

# View Results
res1
#    MODEL  TYPE DEPVAR     RMSE Intercept    speed dist
# 1 MODEL1 PARMS   dist 15.37959 -17.57909 3.932409   -1

# Example 2: SAS Model Syntax
res2 <- proc_reg(dat, model = "dist = speed")

# View Results
res2
```

```
#    MODEL  TYPE DEPVAR     RMSE Intercept    speed dist
# 1 MODEL1 PARMS   dist 15.37959 -17.57909 3.932409   -1

# Example 3: Report Output
res3 <- proc_reg(dat, model = dist ~ speed, output = report)

# View Results
res3
# $NObs
#                         LABEL NOBS
# 1 Number of Observations Read   50
# 2 Number of Observations Used   50
#
# $ANOVA
#             LABEL DF    SUMSQ     MEANSQ     FVAL        PROBF
# 1           Model  1 21185.46 21185.4589 89.56711 1.489919e-12
# 2           Error 48 11353.52   236.5317       NA           NA
# 3 Corrected Total 49 32538.98         NA       NA           NA
#
# $FitStatistics
# RMSE DEPMEAN  COEFVAR       RSQ    ADJRSQ
# 1 15.37959   42.98 35.78312 0.6510794 0.6438102
#
# $ParameterEstimates
#        PARM DF        EST    STDERR         T        PROBT
# 1 Intercept  1 -17.579095 6.7584402 -2.601058 1.231882e-02
# 2     speed  1   3.932409 0.4155128  9.463990 1.489919e-12

# Example 4: By variable
res4 <- proc_reg(dat, model = dist ~ speed, by = group)

# View Results
res4
#        BY  MODEL  TYPE DEPVAR     RMSE  Intercept    speed dist
# 1 Group A MODEL1 PARMS   dist 15.35049 -24.888326 4.275357   -1
# 2 Group B MODEL1 PARMS   dist 15.53676  -8.705547 3.484381   -1

# Example 5: "tableout" Option
res5 <- proc_reg(dat, model = dist ~ speed, options = tableout)

# View Results
res5
#    MODEL   TYPE DEPVAR     RMSE    Intercept        speed dist
# 1 MODEL1  PARMS   dist 15.37959 -17.57909489 3.932409e+00   -1
# 2 MODEL1 STDERR   dist 15.37959   6.75844017 4.155128e-01   NA
# 3 MODEL1      T   dist 15.37959  -2.60105800 9.463990e+00   NA
# 4 MODEL1 PVALUE   dist 15.37959   0.01231882 1.489919e-12   NA
# 5 MODEL1   L95B   dist 15.37959 -31.16784960 3.096964e+00   NA
# 6 MODEL1   U95B   dist 15.37959  -3.99034018 4.767853e+00   NA

# Example 6: Multiple Models plus Statistics Keywords
res6 <- proc_reg(dat, model = list(mod1 = dist ~ speed,
                                   mod2 = speed ~ dist),
```

```
                stats = v(press, seb))

# View Results
res6
#  MODEL  TYPE DEPVAR      RMSE      PRESS   Intercept      speed        dist
# 1 mod1 PARMS   dist 15.379587 12320.2708 -17.5790949  3.9324088 -1.00000000
# 2 mod1   SEB   dist 15.379587         NA   6.7584402  0.4155128 -1.00000000
# 3 mod2 PARMS  speed  3.155753   526.2665   8.2839056 -1.0000000  0.16556757
# 4 mod2   SEB  speed  3.155753         NA   0.8743845 -1.0000000  0.01749448

# Example 6: Plot requests via a vector of keywords
res7 <- proc_reg(dat, model = dist ~ speed,
                      plots = c("residualhistogram", "fitplot", "qqplot", "cooksd"),
                      output = report)

# View results
res7
# $NObs
# LABEL NOBS
# 1 Number of Observations Read   50
# 2 Number of Observations Used   50
#
# $ANOVA
# LABEL DF    SUMSQ     MEANSQ     FVAL        PROBF
# 1           Model  1 21185.46 21185.4589 89.56711 1.489919e-12
# 2           Error 48 11353.52   236.5317       NA           NA
# 3 Corrected Total 49 32538.98         NA       NA           NA
#
# $FitStatistics
# RMSE DEPMEAN  COEFVAR       RSQ    ADJRSQ
# 1 15.37959   42.98 35.78312 0.6510794 0.6438102
#
# $ParameterEstimates
# PARM DF        EST    STDERR         T        PROBT
# 1 Intercept  1 -17.579095 6.7584402 -2.601058 1.231882e-02
# 2     speed  1   3.932409 0.4155128  9.463990 1.489919e-12
#
# $Plots
# $Plots$residualhistogram
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa81dfb8e6.jpg'
# - height: 4.5
# - width: 6
#
# $Plots$fitplot
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa8361465ed.jpg'
# - height: 4.5
# - width: 6
#
# $Plots$qqplot
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa86ca52064.jpg'
```

```
# - height: 4.5
# - width: 6
#
# $Plots$cooksd
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa84d614ded.jpg'
# - height: 4.5
# - width: 6
```

`proc_sort` *Sorts a dataset*

Description

The `proc_sort` function sorts a dataset according to the variables passed on the `by` parameter. If no parameters are passed on the `by` parameter, it will sort by all variables. The direction of the sort is controlled with the `order` parameter. Use the `nodupkey` option to eliminate duplicate rows from the dataset, and the `keep` parameter to subset columns. The parameters will accept either quoted or unquoted values.

Usage

```
proc_sort(
  data,
  by = NULL,
  keep = NULL,
  order = "ascending",
  options = NULL,
  as.character = FALSE,
  na.sort = NULL,
  where = NULL
)
```

Arguments

`data`	The input data to sort.
`by`	A vector of variables to sort by.
`keep`	A vector of variables on the output data to keep. All other variables will be dropped.
`order`	The sort order of the variables on the by parameter. Valid values are 'ascending' or 'descending'. These values may also be abbreviated to 'asc', 'desc', 'a', or 'd'. You may pass a vector of order values equal to the number of variables on the by parameter. Default is 'ascending' for all by variables.
`options`	Any options desired for the sort. Available options are 'dupkey' and 'nodupkey'. The 'nodupkey' option removes duplicate rows from the sorted dataset. The 'dupkey' option removes unique rows from the sorted dataset.

`as.character` If TRUE, will cast any factors in the 'by' parameter to character. Default is FALSE. This parameter is included because it is common to use factors for sorting in R, but you may not want to keep the variable as a factor. This parameter therefore allows you to use the factor for the sort, but then convert back to a character once the sort is complete.

`na.sort` An option that determines how to sort NA values. Valid values are NULL, "first", "last", and "sas". Values may be quoted or unquoted. The "sas" value will sort NAs first for ascending keys, and last for descending keys. The NA sort style may be controlled globally with `options("procs.na.sort" = <value>)`. Any values passed on the local parameter will override the global setting. The default value is NULL, which defers to the global setting. If the global setting is also NULL, the parameter will default to "last".

`where` An expression to filter the rows before the sort takes place. Use the `expression` function to define the where clause.

Value

The sorted dataset. If a data frame was input, a data frame will be output. If a tibble was input, a tibble will be output.

Options

Below are the available options for the `proc_sort` function:

- **dupkey**: This option keeps duplicate rows and discards unique rows. Duplicate rows will be identified by the key variables listed on the by parameter if passed. This option is the opposite of 'nodupkey'.
- **nodupkey**: Removes duplicate rows following the sort. Duplicate rows will be identified by the key variables listed on the by parameter if passed. Otherwise, the function will dedupe on all variables returned.

Missing Values

Missing values are handled in R differently than they are handled in SAS. In R, missing values (NA) are sorted last, and most sorting functions in R give you an option to sort them first if desired.

SAS, on the other hand, considers a missing as the smallest value. If the sort is ascending, it will sort the missing values first. If the sort is descending, it will sort the missing values last.

The `na.sort` parameter on `proc_sort` gives you a choice of how NAs are sorted. By default, the function sorts in the standard R manner. If you are trying to replicate a SAS sort, set the value to "sas".

Note that this parameter only sorts NA values in a manner similar to SAS. It does not guarantee that the entire sort will match SAS. There can still be differences in the sort caused by formats, character-set mismatch, or other reasons.

If you want to always use the SAS style sort, you can set it as a global option, like this: `options("procs.na.sort" = "sas")`.

Examples

```
# Prepare data subset
dat <- data.frame(HairEyeColor, stringsAsFactors = FALSE)[1:32 %% 4 == 1, ]

# View data
dat
#     Hair   Eye    Sex Freq
# 1  Black Brown   Male   32
# 5  Black  Blue   Male   11
# 9  Black Hazel   Male   10
# 13 Black Green   Male    3
# 17 Black Brown Female   36
# 21 Black  Blue Female    9
# 25 Black Hazel Female    5
# 29 Black Green Female    2

# Sort by Frequency
res1 <- proc_sort(dat, by = Freq)

# View results
res1
#    Hair   Eye    Sex Freq
# 29 Black Green Female    2
# 13 Black Green   Male    3
# 25 Black Hazel Female    5
# 21 Black  Blue Female    9
# 9  Black Hazel   Male   10
# 5  Black  Blue   Male   11
# 1  Black Brown   Male   32
# 17 Black Brown Female   36

# Sort by Frequency descending
res2 <- proc_sort(dat, by = Freq, order = d)

# View results
res2
#     Hair   Eye    Sex Freq
# 17 Black Brown Female   36
# 1  Black Brown   Male   32
# 5  Black  Blue   Male   11
# 9  Black Hazel   Male   10
# 21 Black  Blue Female    9
# 25 Black Hazel Female    5
# 13 Black Green   Male    3
# 29 Black Green Female    2

# Get unique combinations of Eye and Sex
res3 <- proc_sort(dat, keep = v(Eye, Sex), options = nodupkey)

# View results
res3
#      Eye    Sex
```

```
# 1  Brown   Male
# 17 Brown Female
# 5   Blue   Male
# 21  Blue Female
# 9  Hazel   Male
# 25 Hazel Female
# 13 Green   Male
# 29 Green Female
```

`proc_transpose` *Transposes a Dataset*

Description

A function to pivot or transpose a data frame. In the default usage, the variables identified by the parameter `var` are transposed to become rows. The variable values in the parameter `id` become the new columns. The function has several more parameters to control how variables are named in the transposed data set. Parameters will accept quoted or unquoted values.

Usage

```
proc_transpose(
  data,
  by = NULL,
  var = NULL,
  id = NULL,
  idlabel = NULL,
  copy = NULL,
  name = "NAME",
  namelabel = NULL,
  prefix = NULL,
  delimiter = ".",
  suffix = NULL,
  where = NULL,
  options = NULL,
  log = TRUE
)
```

Arguments

`data`	The input data to transpose.
`by`	An optional by group. Parameter accepts a vector of one or more quoted variable names. If the by group is requested, the data will be subset by that variable and the transpose function will transpose each group and stack them together in a single table.
`var`	The variable or variables to transpose. Parameter accepts a vector of variable names. By default, all numeric variables will be transposed.

id	The variable or variables to use for the transposed column names.
idlabel	The variable to use for the transposed column labels.
copy	A vector of variables to retain in the output data without transposition. Values will be truncated or recycled to fit the number of output rows.
name	Specifies the name of the variable to be used for the var values.
namelabel	The label to use for the name variable.
prefix	Contains a prefix to be used in the construction of column names.
delimiter	Specifies a delimiter to be used in the construction of column names.
suffix	Contains a suffix to be used in the construction of column names.
where	An expression to filter the rows after the transform is complete. Use the expression function to define the where clause.
options	Optional keywords that affect the transpose. Default is NULL. Available option is "noname" which drops the name column from the output dataset.
log	Whether or not to log the procedure. Default is TRUE. This parameter is used internally.

Details

The proc_tranpose function takes an input data frame or tibble and transposes the columns and rows. If no parameters are specified, the function will assign all numeric variables to the var parameter. These variables will become rows, and generic column names ("COL1", "COL2", etc.) will be generated. Other variables will be dropped.

There are several parameters to control how new column names are constructed. If the desired column names already exist in your data, identify them on the id parameter. The function will then use those data values unaltered. The label for these new columns can also be constructed from data values using the idlabel parameter.

The name and namelabel parameter are used to control the name of the column created for the var values. If this parameter is not passed, the column will be called "NAME", and no label will be assigned.

You may group the transposed values using the by parameter. This parameter accepts one or more variable names to use for grouping. If this parameter is used, the function will first subset the data by the unique combination of by variables, transpose each subset, and then combine the result into a single output dataset. The by group variables will be named on the output dataset with a generic name ("BY1", "BY2", etc.).

The copy parameter is used to simply copy columns from the input dataset to the transposed dataset. If necessary, these values will be recycled or truncated to fit the number of output rows. Any input variables not included in the var, id, or copy parameter will be dropped.

Once the transpose is complete, you may wish to filter the output data. Filtering can be accomplished using the where parameter. This parameter takes an expression using the expression function. The expression is constructed using standard R logical operators. Variable names do not need to be quoted.

The prefix, delimiter, and suffix parameter are used to control how generic column names are constructed. These parameters are especially useful when there are multiple var variables.

Value

The transposed dataset. If a data frame is input, a data frame will be output. If a tibble is input, a tibble will be output.

Examples

```
# Prepare data
dat <- data.frame(CAT = rownames(USPersonalExpenditure),
                  USPersonalExpenditure, stringsAsFactors = FALSE,
                  row.names = NULL)[1:4, ]

# View data
dat
#                   CAT X1940 X1945 X1950 X1955 X1960
# 1    Food and Tobacco 22.20 44.50 59.60  73.2  86.8
# 2 Household Operation 10.50 15.50 29.00  36.5  46.2
# 3  Medical and Health  3.53  5.76  9.71  14.0  21.1
# 4       Personal Care  1.04  1.98  2.45   3.4   5.4

# Default transpose
tdat1 <- proc_transpose(dat)

# View results
tdat1
#    NAME COL1 COL2  COL3 COL4
# 1 X1940 22.2 10.5  3.53 1.04
# 2 X1945 44.5 15.5  5.76 1.98
# 3 X1950 59.6 29.0  9.71 2.45
# 4 X1955 73.2 36.5 14.00 3.40
# 5 X1960 86.8 46.2 21.10 5.40

# Transpose with ID and Name
tdat2 <-  proc_transpose(dat, id = CAT, name = Year)

# View results
tdat2
#   Year Food and Tobacco Household Operation Medical and Health Personal Care
# 1 X1940             22.2                10.5               3.53          1.04
# 2 X1945             44.5                15.5               5.76          1.98
# 3 X1950             59.6                29.0               9.71          2.45
# 4 X1955             73.2                36.5              14.00          3.40
# 5 X1960             86.8                46.2              21.10          5.40

# Transpose only some of the variables
tdat3 <- proc_transpose(dat, var = v(X1940, X1950, X1960), id = CAT, name = Year)

# View results
tdat3
#    Year Food and Tobacco Household Operation Medical and Health Personal Care
# 1 X1940             22.2                10.5               3.53          1.04
# 2 X1950             59.6                29.0               9.71          2.45
# 3 X1960             86.8                46.2              21.10          5.40
```

```
# By with a where clause
tdat4 <- proc_transpose(dat, by = CAT, name = Year,
                        where = expression(Year %in% c("X1940", "X1950", "X1960")))

# View Results
tdat4
#                    CAT  Year  COL1
# 1     Food and Tobacco X1940 22.20
# 2     Food and Tobacco X1950 59.60
# 3     Food and Tobacco X1960 86.80
# 4  Household Operation X1940 10.50
# 5  Household Operation X1950 29.00
# 6  Household Operation X1960 46.20
# 7   Medical and Health X1940  3.53
# 8   Medical and Health X1950  9.71
# 9   Medical and Health X1960 21.10
# 10       Personal Care X1940  1.04
# 11       Personal Care X1950  2.45
# 12       Personal Care X1960  5.40
```

`proc_ttest` *Calculates T-Test Statistics*

Description

The `proc_ttest` function generates T-Test statistics for selected variables on the input dataset. The variables are identified on the `var` parameter or the `paired` parameter. The function will calculate a standard set of T-Test statistics. Results are displayed in the viewer interactively and returned from the function.

Usage

```
proc_ttest(
  data,
  var = NULL,
  paired = NULL,
  output = NULL,
  by = NULL,
  class = NULL,
  options = NULL,
  titles = NULL,
  order = NULL,
  plots = NULL,
  where = NULL
)
```

Arguments

`data`	The input data frame for which to calculate summary statistics. This parameter is required.
`var`	The variable or variables to be used for hypothesis testing. Pass the variable names in a quoted vector, or an unquoted vector using the `v()` function. If there is only one variable, it may be passed unquoted. If the `class` variable is specified, the function will compare the two groups identified in the class variable. If the `class` variable is not specified, enter the baseline hypothesis value on the "h0" option. Default "h0" value is zero (0).
`paired`	A vector of paired variables to perform a paired T-Test on. Variables should be separated by a star (*). The entire string should be quoted, for example, `paired = "var1 * var2"`. To test multiple pairs, place the pairs in a quoted vector : `paired = c("var1 * var2", "var3 * var4")`. The parameter does not accept parenthesis, hyphens, or any other shortcut syntax.
`output`	Whether or not to return datasets from the function. Valid values are "out", "none", and "report". Default is "out", and will produce dataset output specifically designed for programmatic use. The "none" option will return a NULL instead of a dataset or list of datasets. The "report" keyword returns the datasets from the interactive report, which may be different from the standard output. The output parameter also accepts data shaping keywords "long, "stacked", and "wide". These shaping keywords control the structure of the output data. See the **Data Shaping** section for additional details. Note that multiple output keywords may be passed on a character vector. For example, to produce both a report dataset and a "long" output dataset, use the parameter `output = c("report", "out", "long")`.
`by`	An optional by group. If you specify a by group, the input data will be subset on the by variable(s) prior to performing any statistics.
`class`	The `class` parameter is used to perform a unpaired T-Test between two different groups of the same variable. For example, if you want to test for a significant difference between a control group and a test group, where the control and test groups are in rows identified by a variable "Group". Note that there can only be two different values on the class variable. Also, the analysis is restricted to only one class variable.
`options`	A vector of optional keywords. Valid values are: "alpha =", "h0 =", and "noprint". The "alpha = " option will set the alpha value for confidence limit statistics. The default is 95% (alpha = 0.05). The "h0 = " option sets the baseline hypothesis value for single-variable hypothesis testing. The "noprint" option turns off the interactive report.
`titles`	A vector of one or more titles to use for the report output.
`order`	Determines the order of class variable values in the analysis. Valid values are "data", "formatted", "freq", and "internal". Default is NULL, which corresponds to "internal".
`plots`	A plot request. Valid values are NULL, TRUE, "all", a vector of plot names, or a call to `ttestplot`. The value TRUE will give you a default set of plots appropriate for the test you are performing. The value "all" will give you all

available plots for the test you are performing. To request specific plots, pass a vector of plot names or call ttestplot and pass the desired plot names on the type parameter. See the ttestplot documentation for the available plot names and additional details. If there are multiple variables to test, you may pass one plot request which applies to all tests, or multiple plot requests in a list. In the latter case, the plot requests should align one to one with the test variables. The default is NULL, meaning no plots will be generated.

where — An expression to filter the rows before the T-test takes place. Use the expression function to define the where clause.

Details

The proc_ttest function is for performing hypothesis testing. Data is passed in on the data parameter. The function can segregate data into groups using the by parameter. There are also options to determine whether and what results are returned.

The proc_ttest function allows for three types of analysis:

- **One Sample**: The one sample test allows you to perform significance testing of a single variable against a known baseline value or null hypothesis. To perform this test, pass the variable name on the var parameter and the baseline on the h0= option. The one sample T-Test performs a classic Student's T-Test and assumes your data has a normal distribution.
- **Paired Comparison**: The paired comparison is for tests of two variables with a natural pairing and the same number of observations for both measures. For instance, if you are checking for a change in blood pressure for the same group of patients at different time points. To perform a paired comparison, use the paired parameter with the two variables separated by a star (*). The paired T-Test performs a classic Student's T-Test and assumes your data has a normal distribution.
- **Two Independant Samples**: The analysis of two independent samples is used when there is no natural pairing, and there may be a different number of observations in each group. This method is used, for example, if you are comparing the effectiveness of a treatment between two different groups of patients. The function assumes that there is a single variable that contains the analysis values for both groups, and another variable to identify the groups. To perform this analysis, pass the target variable name on the var parameter, and the grouping variable on the class parameter. The Two Sample T-Test provides both a Student's T-Test and a Welch-Satterthwaite T-Test. Select the appropriate T-Test results for your data based on the known normality.

Value

Normally, the requested T-Test statistics are shown interactively in the viewer, and output results are returned as a list of data frames. You may then access individual datasets from the list using dollar sign ($) syntax. The interactive report can be turned off using the "noprint" option, and the output datasets can be turned off using the "none" keyword on the output parameter.

Interactive Output

By default, proc_ttest results will be sent to the viewer as an HTML report. This functionality makes it easy to get a quick analysis of your data.

The `titles` parameter allows you to set one or more titles for your report. Pass these titles as a vector of strings.

The exact datasets used for the interactive report can be returned as a list. To return these datasets, pass the "report" keyword on the `output` parameter. This list may in turn be passed to `proc_print` to write the report to a file.

To turn off the interactive report, pass the "noprint" keyword to the `options` parameter. If you want to turn off the interactive report for all procedures, set the global option `options("procs.print" = FALSE)`.

Dataset Output

Dataset results are also returned from the function by default. `proc_ttest` typically returns multiple datasets in a list. Each dataset will be named according to the category of statistical results. There are three standard categories: "Statistics", "ConfLimits", and "TTests". For the class style analysis, the function also returns a dataset called "Equality" that shows the Folded F analysis.

The output datasets generated are optimized for data manipulation. The column names have been standardized, and additional variables may be present to help with data manipulation. For example, the by variable will always be named "BY". In addition, data values in the output datasets are intentionally not rounded or formatted to give you the most accurate numeric results.

Options

The `proc_ttest` function recognizes the following options. Options may be passed as a quoted vector of strings, or an unquoted vector using the `v()` function.

- **alpha =** : The "alpha = " option will set the alpha value for confidence limit statistics. Set the alpha as a decimal value between 0 and 1. For example, you can set a 90% confidence limit as `alpha = 0.1`.
- **h0**: The "h0 =" option is used to set the baseline mean value for testing a single variable. Pass the option as a name/value pair, such as `h0 = 95`.
- **noprint**: Whether to print the interactive report to the viewer. By default, the report is printed to the viewer. The "noprint" option will inhibit printing. You may inhibit printing globally by setting the package print option to false: `options("procs.print" = FALSE)`.

Class Order

For analysis of two independent samples, the order of the class values can effect the statistical results. The `order` parameter allows you to control that order. The parameter takes four possible values:

- **internal**: The internal or "natural" order of the class values. For character variables, "internal" means alphabetical order. For factors, "internal" means by order of the levels.
- **data**: "data" order is different from "internal" order. "data" order means the order that the data appears in the incoming data frame, which may or may not be the same as the "internal" order. This choice essentially sorts the class values according to however the class variable values were encountered in the incoming data.
- **freq**: The "freq" order value means to sort the class values in descending frequency order.

- **formatted**: If there is a format assigned to the class variable, you may also use "formatted" order. "formatted" order means to apply the format to the class variable and then sort the analysis according to the order specified in the format. The format should be a user-defined format as defined in the **fmtr** package.

Note that the `order` parameter applies to both the interactive report and the data frame output.

Data Shaping

The output datasets produced by the function can be shaped in different ways. These shaping options allow you to decide whether the data should be returned long and skinny, or short and wide. The shaping options can reduce the amount of data manipulation necessary to get the data into the desired form. The shaping options are as follows:

- **long**: Transposes the output datasets so that statistics are in rows and variables are in columns.
- **stacked**: Requests that output datasets be returned in "stacked" form, such that both statistics and variables are in rows.
- **wide**: Requests that output datasets be returned in "wide" form, such that statistics are across the top in columns, and variables are in rows. This shaping option is the default.

These shaping options are passed on the `output` parameter. For example, to return the data in "long" form, use `output = "long"`.

Plots

In addition to T-test statistics, you may request that plots be displayed on the interactive report. The simplest way to request plots is to pass the value TRUE on the `plots` parameters. This action will produce default plots appropriate for the type of analysis you are performing. You may also pass a vector of plot type strings. The available plot types are "agreement", "boxplot", "histogram", "interval", "profiles", "qqplot", and "summary". See the documentation for `ttestplot` for more information.

See Also

`ttestplot()`

Examples

```
# Turn off printing for CRAN checks
options("procs.print" = FALSE)

# Prepare sample data
dat1 <- subset(sleep, group == 1, c("ID", "extra"))
dat2 <- subset(sleep, group == 2, c("ID", "extra"))
dat <- data.frame(ID = dat1$ID, group1 = dat1$extra, group2 = dat2$extra)

# View sample data
dat
#    ID group1 group2
# 1   1    0.7    1.9
# 2   2   -1.6    0.8
```

```
# 3   3   -0.2    1.1
# 4   4   -1.2    0.1
# 5   5   -0.1   -0.1
# 6   6    3.4    4.4
# 7   7    3.7    5.5
# 8   8    0.8    1.6
# 9   9    0.0    4.6
# 10 10    2.0    3.4

# Example 1:  T-Test using h0 option
res1 <- proc_ttest(dat, var = "group1", options = c("h0" = 0))

# View results
res1
# $Statistics
#      VAR  N MEAN     STD    STDERR  MIN MAX
# 1 group1 10 0.75 1.78901 0.5657345 -1.6 3.7
#
# $ConfLimits
#      VAR MEAN       LCLM    UCLM     STD
# 1 group1 0.75 -0.5297804 2.02978 1.78901
#
# $TTests
#      VAR DF       T     PROBT
# 1 group1  9 1.32571 0.2175978

# Example 2: T-Test using paired parameter
res2 <- proc_ttest(dat, paired = "group2 * group1")

# View results
res2
# $Statistics
#     VAR1   VAR2          DIFF  N MEAN      STD    STDERR MIN MAX
# 1 group2 group1 group2-group1 10 1.58 1.229995 0.3889587   0 4.6
#
# $ConfLimits
#     VAR1   VAR2          DIFF MEAN      LCLM     UCLM      STD   LCLMSTD  UCLMSTD
# 1 group2 group1 group2-group1 1.58 0.7001142 2.459886 1.229995 0.8460342 2.245492
#
# $TTests
#     VAR1   VAR2          DIFF DF        T      PROBT
# 1 group2 group1 group2-group1  9 4.062128 0.00283289

# Example 3: T-Test using class parameter
res3 <- proc_ttest(sleep, var = "extra", class = "group")

# View results
res3
# $Statistics
#     VAR      CLASS        METHOD  N  MEAN      STD    STDERR  MIN MAX
# 1 extra          1          <NA> 10  0.75 1.789010 0.5657345 -1.6 3.7
# 2 extra          2          <NA> 10  2.33 2.002249 0.6331666 -0.1 5.5
# 3 extra Diff (1-2)        Pooled NA -1.58       NA 0.8490910   NA  NA
```

```
# 4 extra Diff (1-2) Satterthwaite NA -1.58       NA 0.8490910   NA  NA
#
# $ConfLimits
#     VAR      CLASS        METHOD  MEAN       LCLM      UCLM      STD  LCLMSTD  UCLMSTD
# 1 extra          1          <NA>  0.75 -0.5297804 2.0297804 1.789010 1.230544 3.266034
# 2 extra          2          <NA>  2.33  0.8976775 3.7623225 2.002249 1.377217 3.655326
# 3 extra Diff (1-2)        Pooled -1.58 -3.3638740 0.2038740       NA       NA       NA
# 4 extra Diff (1-2) Satterthwaite -1.58 -3.3654832 0.2054832       NA       NA       NA
#
# $TTests
#     VAR        METHOD VARIANCES       DF         T      PROBT
# 1 extra        Pooled     Equal 18.00000 -1.860813 0.07918671
# 2 extra Satterthwaite   Unequal 17.77647 -1.860813 0.07939414
#
# $Equality
#    VAR   METHOD NDF DDF      FVAL     PROBF
# 1 extra Folded F   9   9 1.252595 0.7427199

# Example 4: T-Test using alpha option and by variable
res4 <- proc_ttest(sleep, var = "extra", by = "group", options = c(alpha = 0.1))

# View results
res4
# $Statistics
#   BY   VAR  N MEAN      STD    STDERR  MIN MAX
# 1  1 extra 10 0.75 1.789010 0.5657345 -1.6 3.7
# 2  2 extra 10 2.33 2.002249 0.6331666 -0.1 5.5
#
# $ConfLimits
# BY   VAR MEAN       LCLM     UCLM      STD  LCLMSTD  UCLMSTD
# 1  1 extra 0.75 -0.2870553 1.787055 1.789010 1.304809 2.943274
# 2  2 extra 2.33  1.1693340 3.490666 2.002249 1.460334 3.294095
#
# $TTests
#   BY   VAR DF        T       PROBT
# 1  1 extra  9 1.325710 0.217597780
# 2  2 extra  9 3.679916 0.005076133

# Example 5: Single variable T-Test using "long" shaping option
res5 <- proc_ttest(sleep, var = "extra", output = "long")

# View results
res5
# $Statistics
#     STAT      extra
# 1      N 20.0000000
# 2   MEAN  1.5400000
# 3    STD  2.0179197
# 4 STDERR  0.4512206
# 5    MIN -1.6000000
# 6    MAX  5.5000000
#
# $ConfLimits
```

```
#      STAT     extra
# 1    MEAN 1.5400000
# 2    LCLM 0.5955845
# 3    UCLM 2.4844155
# 4     STD 2.0179197
# 5 LCLMSTD 1.5346086
# 6 UCLMSTD 2.9473163
#
# $TTests
#    STAT       extra
# 1    DF 19.00000000
# 2     T  3.41296500
# 3 PROBT  0.00291762

# Example 6: Plots
res6 <- proc_ttest(dat, var = "group1", options = c("h0" = 0),
                   output = report,   # To return plot objects
                   plots = TRUE)      # Request default plots

# View results
res6
# $Statistics
# N MEAN     STD    STDERR  MIN MAX
# 1 10 0.75 1.78901 0.5657345 -1.6 3.7
#
# $ConfLimits
# MEAN       LCLM    UCLM     STD  LCLMSTD  UCLMSTD
# 1 0.75 -0.5297804 2.02978 1.78901 1.230544 3.266034
#
# $TTests
# DF       T     PROBT
# 1  9 1.32571 0.2175978
#
# $Plots
# $Plots$summary
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa866215cf2.jpg'
# - height: 4.5
# - width: 6
#
# $Plots$qqplot
# # A plot specification:
# - path: 'C:\Users\dbosa\AppData\Local\Temp\RtmpOYrg1Y\file5dfa84693581d.jpg'
# - height: 4.5
# - width: 6

# Example 7: Order and Where expression
res7 <- proc_ttest(sleep, var = "extra", class = "group",
                   order = "freq",
                   where = expression(extra > 0))
# View results
res7$Statistics
#     VAR      CLASS        METHOD  N MEAN      STD    STDERR MIN MAX
```

```
# 1 extra          2          <NA>  9 2.60 1.920937 0.6403124 0.1 5.5
# 2 extra          1          <NA>  5 2.12 1.406058 0.6288084 0.7 3.7
# 3 extra Diff (1-2)        Pooled NA 0.48       NA 0.9850663  NA  NA
# 4 extra Diff (1-2) Satterthwaite NA 0.48       NA 0.8974408  NA  NA
```

`regplot` *Request Regression Plots*

Description

A function to request regression plots on a call to `proc_reg`. The function allows you to specify the type of regression plots to produce. It produces a combined diagnostics panel, residuals dot plot, and a fit plot by default. You may also specify individual plots by passing a vector of plot names on the "type" parameter.

Usage

```
regplot(
  type = c("diagnostics", "residuals", "fitplot"),
  panel = TRUE,
  stats = "default",
  label = FALSE,
  id = NULL
)
```

Arguments

`type`	The type(s) of plot to create. Multiple types should be passed as a vector of strings. Valid values are "diagnostics", "cooksd", "dfbetas", "dffits", "fitplot", "observedbypredicted", "qqplot", "residuals", "residualboxplot", "residualbypredicted", "residualhistogram", "rfplot", "rstudentbyleverage", and "rstudentbypredicted". The default value is a vector with "diagnostics", "residuals", and "fitplot". The "diagnostics" keyword produces a single combined chart with 8 different plots and a selection of statistics in a small table. The statistics can be controlled by the `stats` parameter. You may also pass the "all" keyword to produce all charts available for the analysis.
`panel`	Whether or not to display the diagnostics plots combined into in a single panel. Default is TRUE. A value of FALSE will create individual plots instead. This parameter is equivalent to the "unpack" keyword in SAS.
`stats`	The statistics to display on the diagnostics panel or fit plot. Valid values are: "adjrsq", "aic", "coeffvar", "depmean", "default", "edf", "mse", "nobs", "nparm", "rsquare", and "sse". The default value is "default", which produces the following statistics: "nobs", "nparm", "edf", "mse", "rsquare", and "adjrsq". You may also pass the value "none" if you do not want any statistics shown on the chart. In the case of the diagnostics panel, if the statistics table is removed, it will be replaced with a residual box plot.

`label` Whether or not to label outlier values. Valid values are TRUE or FALSE. Default is FALSE. If TRUE, this option will assign labels to outlier values on some charts. Charts that support labels are as follows: "diagnostics", "cooksd", "dffits", "dfbetas", "rstudentbypredicted", and "rstudentbyleverage".

`id` If the `label` parameter is TRUE, this parameter determines which value is assigned to the label. By default, the row number will be assigned. You may also assign a column name from the input dataset to this parameter to use as the label value.

Details

There are many types of regression plots. The plots have different uses. Some of the plots help you assess normality of the data. Other plots help you assess the quality of the model. You can select the type of plots you want by passing a vector of plot names on the `type` parameter.

Here is a list of possible plot types and a short description of each:

- **diagnostics**: A fit diagnostics panel that contains 8 different types of plots and a table of statistics.
- **cooksd**: Cook's D statistic vs. Observation number.
- **dfbetas**: Displays the influence of each observation for each coefficient in the model.
- **dffits**: Displays the influence of each observation on fitted values.
- **fitplot**: Produces a scatter plot of the dependent variable against the regressor, including the fitted line and confidence/prediction bands. The fitplot is only available for models with a single regressor.
- **observedbypredicted**: Dependent variable (Observed) vs. Predicted values.
- **qqplot**: Normal Quantile-Quantile (Q-Q) plot of residuals.
- **residuals**: Produces a panel of residual plots against each independent variable in the model.
- **residualboxplot**: A box plot of residual values.
- **residualbypredicted**: Residuals vs. Predicted values.
- **residualhistogram**: Histogram of residuals, with a normal and kernel curve overlay.
- **rfplot**: Residual-Fit (RF) spread plot.
- **rstudentbyleverage**: Externally Studentized Residuals vs. Leverage.
- **rstudentbypredicted**: Externally Studentized Residuals (RStudent) vs. Predicted values.

If possible, the statistics from the tabular report are used for the plots. Otherwise, additional statistics functions are called to produce the needed values.

Plot Statistics

The diagnostics panel and the fit plot each contain a small table of statistics. This table is customizable. The following keywords may be used to to customize the statistics table:

- **adjrsq**: Adjusted R-square.
- **aic**: Akaike's information criterion.
- **coeffvar**: Coefficient of variation.

- **depmean**: Mean of dependent.
- **default**: A set of default statistics.
- **edf**: Error degrees of freedom.
- **mse**: Mean squared error.
- **nobs**: Number of observations used.
- **nparm**: Number of parameters in the model (including the intercept).
- **rsquare**: The R-square statistic.
- **sse**: Error sum of squares.

To use these keywords, pass them as a vector to the `stats` parameter on the `regplot` function.

Labeling Outliers

Some types of plots can be used to identify outliers in the data. For instance, the Cook's D chart is excellent for such a task. When identifying outliers, it is helpful to have them labelled on the chart. The labels make it possible to trace the outliers back to the source data.

To get a basic row/observation label, set the `labels` parameter to TRUE. If there is a column in the data that can be used to identify an individual record, pass that column name on the `id` parameter. The values from that column will then be used as outlier labels on those charts that support labels.

Additional Information

Regression plots will be displayed on interactive reports only. Plots are created as jpeg files, and stored in a temp directory. Those temporary files are then referenced by the interactive report to display the graphic.

If desired, you may output the report objects and pass to `proc_print`. To do this, set `output = report` on the call to `proc_reg`, and pass the entire list to `proc_print`.

Discrepancies with SAS

The histogram binning algorithm in R is different from the binning algorithm in SAS®. R uses a "Sturges" algorithm, which more accurately represents the distribution of the data. This algorithm may produce a different number of bins and different height of bars than the corresponding SAS chart.

Plots generated by `proc_reg` may also have some spacing, color, and shape discrepancies with the corresponding SAS charts. The information conveyed is expected to be similar.

See Also

`proc_reg()`

Examples

```
library(procs)

# Turn off printing for CRAN checks
# Set to TRUE to run in local environment
```

```
options("procs.print" = FALSE)


# Example 1: Regression statistics with default plots
res <- proc_reg(iris, model = "Sepal.Length = Petal.Length",
                output = report,
                plots = TRUE,
                titles = "Iris Regression Statistics")

# View results
res

# Example 2: Regression statistics with custom plot request and by variable
res <- proc_reg(iris, model = "Sepal.Length = Petal.Length",
                output = report,
                by = Species,
                plots = regplot(type = v(residualhistogram,
                                rstudentbypredicted, rstudentbyleverage),
                                label = TRUE),
                titles = "Iris Regression Statistics")

# View results
res

# Example 3: Regression statistics with multiple models, same plot string
res <- proc_reg(iris, model = c("Sepal.Length = Petal.Length",
                                "Sepal.Length = Sepal.Width",
                                "Sepal.Length = Petal.Width"),
                output = report,
                plots = "diagnostics",
                titles = "Iris Regression Statistics")

# View results
res

# Example 4: Regression statistics with multiple models, different plot strings
res <- proc_reg(iris, model = c("Sepal.Length = Petal.Length",
                                "Sepal.Length = Sepal.Width",
                                "Sepal.Length = Petal.Width"),
                output = report,
                plots = list("diagnostics",
                             "residualhistogram",
                             "fitplot"),
                titles = "Iris Regression Statistics")

# View results
res

# Example 5: Regression statistics with multiple models, different plot functions
res <- proc_reg(iris, model = c("Sepal.Length = Petal.Length",
                                "Sepal.Length = Sepal.Width",
                                "Sepal.Length = Petal.Width"),
                output = report,
```

```
                  plots = list(regplot(type = "diagnostics"),
                               regplot(type = "cooksd",
                                       label = TRUE),
                               regplot(type = "fitplot",
                                       stats = c("nobs", "mse", "rsquare"))),
                  titles = "Iris Regression Statistics")

# View results
res

# Example 6: Regression statistics with multiple models, influence charts
res <- proc_reg(iris, model = c("Sepal.Length = Petal.Length",
                                "Sepal.Length = Petal.Length Petal.Width",
                                "Sepal.Length = Petal.Length Petal.Width Sepal.Width"),
                output = report,
                plots = regplot(v(cooksd, dffits, dfbetas), label = TRUE),
                titles = "Iris Regression Statistics")

# View results
res
```

`ttestplot` *Request T-Test Plots*

Description

A function to request T-test plots on a call to `proc_ttest`. The function allows you to specify the type of T-Test plots to produce. By default, it produces plots appropriate to the analysis being performed. You may also request specific plots by passing a vector of plot names to the `type` parameter.

Usage

```
ttestplot(
  type = "default",
  panel = TRUE,
  showh0 = FALSE,
  label = TRUE,
  id = NULL
)
```

Arguments

`type`	The type(s) of plot to create. Multiple types should be passed as a vector of strings. Valid values are "agreement", "boxplot", "histogram", "interval", "profiles", "qqplot", and "summary". The default value is "default", which will produce default plots appropriate for the analysis. You may use the `v` function to pass the plot keywords unquoted.

panel	Whether or not to display the summary plot combined into in a single panel. Default is TRUE. A value of FALSE will create individual "histogram" and "boxplot" charts instead. This parameter is equivalent to the "unpack" keyword in SAS.
showh0	Whether or not to show the null hypothesis on the chart. Default is FALSE. If this parameter is TRUE, a vertical line will be created on some charts to show the value specified for the "h0=" option on proc_ttest.
label	Whether or not to label outlier values. Valid values are TRUE or FALSE. Default is TRUE. If TRUE, this option will assign labels to outlier values on the "summary" and "boxplot" charts.
id	If the label parameter is TRUE, this parameter determines which value is assigned to the label. By default, the row number will be assigned. You may also assign a column name from the input dataset to this parameter to use as the label value.

Details

Plots produced by proc_ttest can change depending on the type of analysis being performed.

For one sample T-Tests, the function creates single variable charts. The "summary" and "qqplot" charts are generated by default. For this type of analysis, it is helpful to set the "showh0" parameter to TRUE so you can see the null hypothesis on the chart.

For two sample T-Tests, the function produces two variable charts. For example, on a two sample T-Test, the "histogram" chart will show histograms for each class value on the same plot. Likewise, the "boxplot" and "qqplot" are separated for each variable. The "interval" plot is split into "Satterthwaite" and "Pooled" intervals. Note that the "h0" option also produces a null hypothesis line for the "interval" plot, if requested.

For a paired analysis, the "profiles" and "agreement" charts are available in addition to the "histogram", "boxplot", "interval", and "qqplot" charts. Paired analysis also produces single variable charts.

To see all plots available for the requested analysis, pass the keyword "all" to the type parameter.

Any requested plots will be displayed on interactive reports only. Plots are created as jpeg files, and stored in a temp directory. Those temporary files are then referenced by the interactive report to display the graphic.

If desired, you may output the report objects and pass to proc_print. To do this, set output = report on the call to proc_freq, and pass the entire list to proc_print.

Plots

The type parameter allows you to request several types of T-Test plots. Below are the types of plots that are supported. The list shows the plot type keyword needed to request the plot, and a brief description:

- **agreement**: An agreement plot for the input data. Only available for paired comparisons.
- **boxplot**: Displays a box and whisker plot for group comparisons, including confidence intervals (if appropriate).
- **histogram**: A histogram with normal curve and kernel density overlays.

- **interval**: Visualizes the confidence intervals for means.
- **profiles**: A line plot mapping responses between analysis variables. Available only for paired analysis.
- **qqplot**: A quantile-quantile plot used to assess the assumption of normality.
- **summary**: Combines the histogram and boxplot charts onto the same panel. For two-sample T-Tests, both the histogram and boxplots are separated for each class value.

The above plots may be requested as a vector of keywords to the `plots` parameter on `proc_ttest`, or as vector of values to the `type` parameter on `ttestplots`.

Note that, when passed as a vector of keywords, only the requested plots will be produced. That is to say, the "only" keyword in SAS is implied at all times for `proc_ttest`.

Discrepancies with SAS

The histogram binning algorithm in R is different from the binning algorithm in SAS®. R uses a "Sturges" algorithm, which more accurately represents the distribution of the data. This algorithm may produce a different number of bins and different height of bars than the corresponding SAS chart.

Plots generated by `proc_ttest` may also have some spacing, color, and shape discrepancies with the corresponding SAS charts. The information conveyed is expected to be similar.

See Also

`proc_ttest()`

Examples

```
library(procs)

# Turn off printing for CRAN checks
options("procs.print" = FALSE)

# View sample data
sleep
#    extra group ID
# 1    0.7     1  1
# 2   -1.6     1  2
# 3   -0.2     1  3
# 4   -1.2     1  4
# 5   -0.1     1  5
# 6    3.4     1  6
# 7    3.7     1  7
# 8    0.8     1  8
# 9    0.0     1  9
# 10   2.0     1 10
# 11   1.9     2  1
# 12   0.8     2  2
# 13   1.1     2  3
# 14   0.1     2  4
# 15  -0.1     2  5
```

```
# 16   4.4     2  6
# 17   5.5     2  7
# 18   1.6     2  8
# 19   4.6     2  9
# 20   3.4     2 10


# Example 1: plots = TRUE
proc_ttest(sleep,
           var = "extra",
           class = "group",
           plots = TRUE)

# Example 2: plots = "all"
proc_ttest(sleep,
           var = "extra",
           class = "group",
           plots = "all")

# Example 3: plots = {vector}
proc_ttest(sleep,
           var = "extra",
           class = "group",
           plots = c("histogram", "boxplot", "interval"))

# Example 4: plots = ttestplot()
proc_ttest(sleep,
           var = "extra",
           class = "group",
           plots = ttestplot("summary", panel = FALSE))
```

Index