

Package ‘statuser’

April 25, 2026

Type Package

Title Statistical Tools Designed for End Users

Version 0.2.1

Author Uri Simonsohn [aut, cre]

Maintainer Uri Simonsohn <urisohn@gmail.com>

Description The statistical tools in this package do one of four things:

- 1) Enhance basic statistical functions with more flexible inputs, smarter defaults, and richer, clearer, and ready-to-use output (e.g., `t.test2()`)
- 2) Produce publication-ready commonly needed figures with one line of code (e.g., `plot_cdf()`)
- 3) Implement novel analytical tools developed by the authors (e.g., `twolines()`)
- 4) Deliver niche functions of high value to the authors that are not easily available elsewhere (e.g., `clear()`, `convert_to_sql()`, `resize_images()`).

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

Imports mgcv, marginaleffects, rsvg, magick, sandwich, lmtest, utils

Suggests testthat (>= 3.0.0), crayon, quantreg, estimatr, broom, modelsummary

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-04-25 12:00:02 UTC

Contents

clear	2
convert_to_sql	3
desc_var	4
format_pvalue	6
interprobe	6

list2	9
lm2	10
message2	13
plot_cdf	14
plot_density	16
plot_freq	18
plot_gam	20
plot_means	22
predict.lm2	24
print.desc_var	25
print.lm2	26
print.t.test2	26
print.table2	27
resize_images	27
scatter.gam	29
summary.lm2	31
t.test2	31
table2	32
text2	34
twolines	35
var_labels	37

Index 38

clear	<i>Clear Plot, Global Environment, and Console</i>
-------	--

Description

Clears plot, global environment, and console. On first use the user is prompted to authorize clearing the environment to comply with CRAN rules.

Usage

```
clear()
```

Details

This function performs three cleanup operations:

- **Plot:** Closes all open graphics devices (except the null device)
- **Global environment:** Removes all objects from the global environment
- **Console:** Clears the console screen (only in interactive sessions)

`clear()` will not modify the global environment unless you have previously typed "yes" when prompted. If you do not type "yes", you are asked again next time; only "yes" is remembered for future sessions.

Warning: This function deletes all objects in the global environment. Save anything that you wish to keep before running.

Value

Invisibly returns NULL. Prints a colored confirmation message.

Examples

```
# Interactive use: clear workspace, console, and plots
# First run may prompt; once you type "yes", your preference is saved.
clear()
```

convert_to_sql	<i>Convert CSV file to SQL INSERT statements</i>
----------------	--

Description

Reads a CSV file and generates SQL statements to insert all rows. Optionally can also generate a CREATE TABLE statement. The function automatically infers column types (REAL for numeric, DATE for date strings matching YYYY-MM-DD format, TEXT otherwise).

Usage

```
convert_to_sql(input, output, create_table = FALSE)
```

Arguments

input	Character string. Path to the input CSV file.
output	Character string. Path to the output SQL file where the statements will be written.
create_table	Logical. If TRUE, includes a CREATE TABLE statement before the INSERT statements. Default is FALSE.

Details

The function performs the following steps:

1. Reads the CSV file using `read.csv()` with `stringsAsFactors = FALSE`
2. Infers SQL column types:
 - Numeric columns become REAL
 - Date columns (matching YYYY-MM-DD format) become DATE
 - All other columns become TEXT
3. If `create_table = TRUE`, generates a CREATE TABLE statement using the base filename (without extension) as the table name
4. Generates INSERT INTO statements for each row
5. Writes all SQL statements to the output file

Single quotes in text values are escaped by doubling them (SQL standard). Numeric values are inserted without quotes, while text and date values are wrapped in single quotes.

Value

Invisibly returns NULL. The function writes SQL statements to the specified output file.

Examples

```
# Convert a CSV file to SQL (INSERT statements only)
tmp_csv <- tempfile(fileext = ".csv")
tmp_sql <- tempfile(fileext = ".sql")
write.csv(
  data.frame(id = 1:2, value = c("a", "b"), date = c("2024-01-01", "2024-02-02")),
  tmp_csv,
  row.names = FALSE
)
convert_to_sql(tmp_csv, tmp_sql)

# Convert a CSV file to SQL with CREATE TABLE statement
convert_to_sql(tmp_csv, tmp_sql, create_table = TRUE)
```

desc_var

Describe a variable, optionally by groups

Description

Returns a dataframe with one row per group.

Usage

```
desc_var(y, group = NULL, data = NULL, digits = 3)
```

Arguments

y	A numeric vector of values, a column name (character string or unquoted) if data is provided, or a formula of the form $y \sim x$ or $y \sim x_1 + x_2$ (for multiple grouping variables).
group	Optional grouping variable, if not provided computed for the full data. Ignored if y is a formula.
data	Optional data frame containing the variable(s).
digits	Number of decimal places to round to. Default is 3.

Details

The dependent variable ('y') must be numeric. If you pass an existing non-numeric variable (e.g., character, factor), 'desc_var()' will stop with a clear error indicating that the variable must be numeric.

Value

A data frame with one row per group (or one row if no group is specified) containing:

- group: Group identifier
- mean: Mean
- sd: Standard deviation
- se: Standard error
- median: Median
- min: Minimum
- max: Maximum
- mode: Most frequent value
- freq_mode: Frequency of mode
- mode2: 2nd most frequent value
- freq_mode2: Frequency of 2nd mode
- n.total: Number of observations
- n.missing: Number of observations with missing (NA) values
- n.unique: Number of unique values

Columns may also carry a "label" attribute, which provides a short human-readable description of each column.

Examples

```
# With grouping
df <- data.frame(y = rnorm(100), group = rep(c("A", "B"), 50))
desc_var(y, group, data = df)

# Without grouping (full dataset)
desc_var(y, data = df)

# Direct vectors
y <- rnorm(100)
group <- rep(c("A", "B"), 50)
desc_var(y, group)

# With custom decimal places
desc_var(y, group, data = df, digits = 2)

# Using formula syntax: y ~ x
desc_var(y ~ group, data = df)

# Using formula syntax with multiple grouping variables: y ~ x1 + x2
df2 <- data.frame(y = rnorm(200), x1 = rep(c("A", "B"), 100), x2 = rep(c("X", "Y"), each = 100))
desc_var(y ~ x1 + x2, data = df2)
```

format_pvalue	<i>Format P-Values for Display</i>
---------------	------------------------------------

Description

Formats p-values for clean display in figures and tables. e.g., $p = .0231$, $p < .0001$

Usage

```
format_pvalue(p, digits = 4, include_p = FALSE)
```

Arguments

p	A numeric vector of p-values to format.
digits	Number of decimal places to round to. Default is 4.
include_p	Logical. If TRUE, includes "p" prefix before the formatted value (e.g., "p = .05"). Default is FALSE.

Value

A character vector of formatted p-values.

Examples

```
# Basic usage
format_pvalue(0.05)
format_pvalue(0.0001)

# More rounding
format_pvalue(0.0001, digits=2)

# Vector input
format_pvalue(c(0.05, 0.001, 0.00001, 0.99))

# With p prefix
format_pvalue(0.05, include_p = TRUE)
```

interprobe	<i>Probe interactions robustly to nonlinearities</i>
------------	--

Description

Probes an interaction by estimating (or accepting) a model and computing: - simple slopes ("spot-lights") using predicted values - Johnson-Neyman ("jn") curves using marginal effects

Usage

```

interprobe(
  x = NULL,
  z = NULL,
  y = NULL,
  model = NULL,
  data = NULL,
  moderator.on.x.axis = TRUE,
  k = NULL,
  spotlights = NULL,
  spotlight.labels = NULL,
  histogram = TRUE,
  max.unique = 11,
  n.bin.continuous = 10,
  n.max = 50,
  xlab = "",
  ylab1 = "",
  ylab2 = "",
  cols = c("red4", "dodgerblue", "green4"),
  main1 = "GAM Simple Slopes",
  main2 = "GAM Johnson-Neyman",
  legend.round = c(1, 4),
  draw = "both",
  save.as = NULL,
  xlim = NULL,
  ylim1 = NULL,
  ylim2 = NULL,
  x.ticks = NULL,
  y1.ticks = NULL,
  y2.ticks = NULL,
  legend.simple.slopes = NULL,
  legend.jn = NULL,
  quiet = FALSE,
  probe.bins = 100
)

```

Arguments

x	The focal predictor. Can be a name (bare or quoted) when ‘data’ or ‘model’ is provided, or a numeric/factor vector when probing from vectors.
z	The moderator. Same options as ‘x’.
y	The dependent variable. Same options as ‘x’. Not required when ‘model’ is supplied.
model	By default ‘interprobe’ estimates a GAM model predicting ‘y’ with ‘x’ and ‘z’. You can instead probe a linear interaction by setting model=linear. You can also probe a model of your choice by running it separately, saving the output, and submitting it as the model argument to interprobe. This is the way to include covariates for a probed interaction.

<code>data</code>	Optional data frame containing 'x', 'z', and 'y'.
<code>moderator.on.x.axis</code>	Logical. If TRUE (default), moderator ('z') is shown on the x-axis.
<code>k</code>	Integer. Smoothness parameter passed to 'mgcv::gam()' when estimating with the default GAM engine.
<code>spotlights</code>	Numeric vector of length 3. Values at which curves are computed.
<code>spotlight.labels</code>	Character vector of length 3. Labels for the legend.
<code>histogram</code>	Logical. If TRUE (default), show sample size distribution under the plot.
<code>max.unique</code>	Integer. Threshold for treating a variable as continuous vs discrete.
<code>n.bin.continuous</code>	Integer. Number of bins used in histogram when binning continuous values.
<code>n.max</code>	Integer. Sample size at which line darkness/width saturates.
<code>xlab</code>	Character. X-axis label.
<code>ylab1</code>	Character. Y-axis label for simple slopes panel.
<code>ylab2</code>	Character. Y-axis label for JN panel.
<code>cols</code>	Character vector of length 3. Colors for the three curves.
<code>main1</code>	Character. Title for simple slopes panel.
<code>main2</code>	Character. Title for JN panel.
<code>legend.round</code>	Integer vector length 2. Min/max decimals in legend.
<code>draw</code>	Which plots to draw: "both" (default), "simple slopes" (or legacy "simple.slopes"), or "jn".
<code>save.as</code>	Optional file path to save plot ('.png' or '.svg').
<code>xlim</code>	Numeric vector length 2. X-axis limits.
<code>ylim1</code>	Numeric vector length 2. Y-axis limits for simple slopes.
<code>ylim2</code>	Numeric vector length 2. Y-axis limits for JN.
<code>x.ticks</code>	Optional custom x-axis ticks.
<code>y1.ticks</code>	Optional custom y-axis ticks for panel 1.
<code>y2.ticks</code>	Optional custom y-axis ticks for panel 2.
<code>legend.simple.slopes</code>	Optional legend title for simple slopes.
<code>legend.jn</code>	Optional legend title for JN.
<code>quiet</code>	Logical. If TRUE, reduces console output.
<code>probe.bins</code>	Integer. Resolution for probing curves (larger = smoother/slower).

Details

Designed for GAM models but works with any model supported by 'marginaleffects' (including 'lm', 'glm', 'mgcv::gam', and 'lm2' / 'estimatr::lm_robust').

Value

Invisibly returns a list with:

- `simple.slopes`: data.frame of predicted values and confidence intervals
- `johnson.neyman`: data.frame of marginal effects and confidence intervals
- `frequencies`: data.frame with bin frequencies used for shading/histogram
- `gam_results`: the fitted GAM model when estimated inside `interprobe()`
- `gam_results_testing`: when `interprobe()` estimates a GAM internally and `x` has exactly 2 unique values, a separate GAM fit used for interaction testing (with a `ti()` term and numeric coding of `x`)
- `lm2_results`: when `interprobe()` estimates a GAM internally, also returns the corresponding linear fit `lm2(y ~ x * z)` (or NULL if package `estimatr` is not installed)

`list2`

Enhanced alternative to `list()`

Description

List with objects that are automatically named.

Usage

```
list2(...)
```

Arguments

... Objects to include in the list. Objects are automatically named based on their variable names unless explicit names are provided.

Details

`list2(x, y)` is equivalent to `list(x = x, y = y)`

`list2(x, y2 = y)` is equivalent to `list(x = x, y2 = y)`

Based on: <https://stackoverflow.com/questions/16951080/can-lists-be-created-that-name-themselves-bas>

Value

A named list. Each element is named after the variable passed to the function (or the explicit name if provided). The structure is identical to a standard R list created with `list`.

Examples

```
x <- 1:5
y <- letters[1:3]
z <- matrix(1:4, nrow = 2)

# Create named list from objects
my_list <- list2(x, y, z)
names(my_list) # "x" "y" "z"

# Works with explicit names too
my_list2 <- list2(a = x, b = y)
names(my_list2) # "a" "b"
```

lm2

Enhanced alternative to lm()

Description

Runs a linear regression with better defaults (robust SE), and richer & better formatted output than `lm`. For robust and clustered errors it relies on `lm_robust`. The output reports classical and robust errors, number of missing observations per variable, an effect size column (standardized regression coefficient), and a `red.flag` column per variable flagging the need to conduct specific diagnostics. It relies by default on HC3 for standard errors; `lm_robust` relies on HC2 (and Stata's `'reg y x, robust'` on HC1), which can have inflated false-positive rates in smaller samples (Long & Ervin, 2000).

Arguments

<code>se_type</code>	The type of standard error to use. Default is "HC3". Without clusters: "HC0", "HC1", "HC2", or "HC3". When <code>clusters</code> is specified, <code>se_type</code> is automatically set to "CR2".
<code>notes</code>	Logical. If TRUE (default), print explanatory notes below the table when the result is printed.
<code>clusters</code>	An optional variable indicating clusters for cluster-robust standard errors. When specified, <code>se_type</code> is automatically set to "CR2" (bias-reduced cluster-robust estimator). Passed to <code>lm_robust</code> .
<code>fixed_effects</code>	An optional right-sided formula containing the fixed effects to be projected out (absorbed) before estimation. Useful for models with many fixed effect groups (e.g., <code>~ firm_id</code> or <code>~ firm_id + year</code>). Passed to <code>lm_robust</code> .
<code>...</code>	Additional arguments passed to <code>lm_robust</code> .

Details

Robust standard errors and clustered standard errors are computed using `lm_robust`; see the documentation of that function for details (using by default CR2 errors) The output shows both standard errors and when clustering errors it reports all three. The `red.flag` column is based on the difference between robust and classical standard errors.

The `red.flag` column provides diagnostic warnings:

- `!`, `!!`, `!!!`: Robust and classical standard errors differ by more than 25%, 50%, or 100%, respectively. Large differences may suggest model misspecification or outliers (but they may also be benign). When encountering a red flag, authors should plot the distributions to look for outliers or skewed data, and use `scatter.gam` to look for possible nonlinearities in the relevant variables. King & Roberts (2015) propose a higher cutoff, at 100%, and a bootstrapped significance test; `statuser` does not follow either recommendation. The former seems too liberal, the latter too time consuming to include in every regression, plus the focus here is on individual variables rather than joint tests.
- `X`: For interaction terms, the component variables are correlated ($|r| > 0.3$ or $p < .05$), which means the interaction term is likely to be biased. See Simonsohn (2024) "Interacting with curves" [doi:10.1177/25152459231207787](https://doi.org/10.1177/25152459231207787).

Value

An object of class `c("lm2", "lm_robust", "lm")`. This inherits from `lm_robust` and can be used with packages like `marginalEffects`. The object contains all components of an `lm_robust` object plus additional attributes:

statuser_table A data frame with columns: `term`, `estimate`, `SE.robust`, `SE.classical`, `t`, `df`, `p.value`, `B` (standardized coefficient), and optionally `SE.cluster` when clustered standard errors are used.

classical_fit The underlying `lm` object with classical standard errors.

na_counts Integer vector of missing value counts per variable.

n_missing Total number of observations excluded due to missing values.

has_clusters Logical indicating whether clustered standard errors were used.

When printed, displays a formatted regression table with robust and classical standard errors, effect sizes, and diagnostic red flags.

References

King, G., & Roberts, M. E. (2015). How robust standard errors expose methodological problems they do not fix, and what to do about it. *Political Analysis*, 23(2), 159-179.

Long, J. S., & Ervin, L. H. (2000). Using heteroscedasticity consistent standard errors in the linear regression model. *The American Statistician*, 54(3), 217-224.

Simonsohn, U. (2024). Interacting with curves: How to validly test and probe interactions in the real (nonlinear) world. *Advances in Methods and Practices in Psychological Science*, 7(1), 1-22.

See Also

[lm_robust](#), [scatter.gam](#)

Examples

```
# Basic usage with data argument
lm2(mpg ~ wt + hp, data = mtcars)

# Without data argument (variables from environment)
```

```

y <- mtcars$mpg
x1 <- mtcars$wt
x2 <- mtcars$hp
lm2(y ~ x1 + x2)

# RED FLAG EXAMPLES

# Example 1: red flag catches a nonlinearity
# True model is quadratic:  $y = x^2$ 
set.seed(123)
x <- runif(200, -3, 3)
y <- x^2 + rnorm(200, sd = 2)

# lm2() shows red flag due to misspecification
lm2(y ~ x)

# Follow up with scatter.gam() to diagnose it
scatter.gam(x, y)

# Example 2: red flag catches an outlier in y
# True model is  $y = x$ , but one observation has a very large y value
set.seed(123)
x <- sort(rnorm(200))
y <- round(x + rnorm(200, sd = 2), 1)
y[200] <- 100 # Outlier

# lm2() flags x
lm2(y ~ x)

# Look at distribution of y to spot the outlier
plot_freq(y)

# Example 3: red flag catches an outlier in one predictor
# True model is  $y = x1 + x2$ , but x2 has an extreme value
set.seed(123)
x1 <- round(rnorm(200), .1)
x2 <- round(rnorm(200), .1)
y <- x1 + x2 + rnorm(200, sd = 0.5)
x2[200] <- 50 # Outlier in x2

# lm2() flags x2 (but not x1)
lm2(y ~ x1 + x2)

# Look at distribution of x2 to spot the outlier
plot_freq(x2)

# CLUSTERED STANDARD ERRORS
# When observations are grouped (e.g., students within schools),
# use clusters to account for within-group correlation
set.seed(123)
n_clusters <- 20
n_per_cluster <- 15
cluster_id <- rep(1:n_clusters, each = n_per_cluster)

```

```

cluster_effect <- rnorm(n_clusters, sd = 2)[cluster_id]
x <- rnorm(n_clusters * n_per_cluster)
y <- 1 + 0.5 * x + cluster_effect + rnorm(n_clusters * n_per_cluster)
mydata <- data.frame(y = y, x = x, cluster_id = cluster_id)

# Clustered SE (CR2) - note the SE.cluster column in output
lm2(y ~ x, data = mydata, clusters = cluster_id)

# FIXED EFFECTS
# Use fixed_effects to absorb group-level variation (e.g., firm or year effects)
# This is useful for panel data or when you have many fixed effect levels
set.seed(456)
n_firms <- 30
n_years <- 5
firm_id <- rep(1:n_firms, each = n_years)
year <- rep(2018:2022, times = n_firms)
firm_effect <- rnorm(n_firms, sd = 3)[firm_id]
x <- rnorm(n_firms * n_years)
y <- 2 + 0.8 * x + firm_effect + rnorm(n_firms * n_years)
panel <- data.frame(y = y, x = x, firm_id = factor(firm_id), year = factor(year))

# Absorb firm fixed effects (coefficient on x is estimated, firm dummies are not shown)
lm2(y ~ x, data = panel, fixed_effects = ~ firm_id)

# Two-way fixed effects (firm and year)
lm2(y ~ x, data = panel, fixed_effects = ~ firm_id + year)

```

message2

Enhanced alternative to message()

Description

Add options to set color and to end execution of code (to be used as error message)

Usage

```
message2(..., col = "cyan", font = 1, stop = FALSE)
```

Arguments

...	Message content to be printed. Multiple arguments are pasted together.
col	text color. Default is "cyan".
font	Integer. 1 for plain text (default), 2 for bold text.
stop	Logical. If TRUE, stops execution (like stop()) but without printing "Error:".

Details

This function prints colored messages to the console. If ANSI color codes are supported by the terminal, the message will be colored. Otherwise, it will be printed as plain text. If `stop = TRUE`, execution will be halted after printing the message.

Value

No return value, called for side effects. Prints a colored message to the console. If `stop = TRUE`, execution is halted after printing the message.

Examples

```
message2("This is a plain cyan message", col = "cyan", font = 1)
message2("This is a bold cyan message", col = "cyan", font = 2)
message2("This is a bold red message", col = "red", font = 2)

cat("this will be shown")
try(message2("This stops execution", stop = TRUE), silent = TRUE)
cat("this will be shown after the try")
```

plot_cdf

Plot Empirical Cumulative Distribution Functions by Group

Description

Plots empirical cumulative distribution functions (ECDFs) separately for each unique value of a grouping variable, with support for vectorized plotting parameters. If no grouping variable is provided, plots a single ECDF.

Usage

```
plot_cdf(
  formula,
  y2 = NULL,
  data = NULL,
  order = NULL,
  show.ks = TRUE,
  show.quantiles = TRUE,
  ...
)
```

Arguments

formula	Two possible uses (similar to <code>t.test()</code>): <ul style="list-style-type: none"> • Single Variable (possibly by subgroup): <code>plot_cdf(y)</code> or <code>plot_cdf(y~x)</code> • Contrast Two Variables: <code>plot_cdf(y1, y2)</code>
y2	optional second variable when contrasting two variables <code>plot_cdf(y1, y2)</code>
data	An optional data frame containing the variables in the formula. If data is not provided, variables are evaluated from the calling environment.
order	Controls the order in which groups appear in the plot and legend. Use <code>-1</code> to reverse the default order. Alternatively, provide a vector specifying the exact order (e.g., <code>c("B", "A", "C")</code>). If <code>NULL</code> (default), groups are ordered by their factor levels (if the grouping variable is a factor) or sorted alphabetically/numerically. Only applies when using grouped plots.
show.ks	Logical. If <code>TRUE</code> (default), shows Kolmogorov-Smirnov test results when there are exactly 2 groups. If <code>FALSE</code> , KS test results are not displayed.
show.quantiles	Logical. If <code>TRUE</code> (default), shows horizontal lines and results at 25th, 50th, and 75th percentiles when there are exactly 2 groups. If <code>FALSE</code> , quantile lines and results are not displayed.
...	Additional arguments passed to plotting functions. Can be single values (applied to all groups) or vectors (applied element-wise to each group). Common parameters include <code>col</code> , <code>lwd</code> , <code>lty</code> , <code>pch</code> , <code>type</code> , etc.

Value

Invisibly returns a list containing:

- `ecdfs`: A list of ECDF function objects, one per group. Each can be called as a function to compute cumulative probabilities (e.g., `result$ecdfs[[1]](5)` returns $P(X \leq 5)$ for group 1).
- `ks_test`: (Only when exactly 2 groups) The Kolmogorov-Smirnov test result comparing the two distributions. Access p-value with `resultks_testp.value`.
- `quantile_regression_25`: (Only when exactly 2 groups) Quantile regression model for the 25th percentile.
- `quantile_regression_50`: (Only when exactly 2 groups) Quantile regression model for the 50th percentile (median).
- `quantile_regression_75`: (Only when exactly 2 groups) Quantile regression model for the 75th percentile.
- `warnings`: Any warnings captured during execution (if any).

Examples

```
# Basic usage with single variable (no grouping)
y <- rnorm(100)
plot_cdf(y)

# Basic usage with formula syntax and grouping
```

```

group <- rep(c("A", "B", "C"), c(30, 40, 30))
plot_cdf(y ~ group)

# With custom colors (scalar - same for all)
plot_cdf(y ~ group, col = "blue")

# With custom colors (vector - different for each group)
plot_cdf(y ~ group, col = c("red", "green", "blue"))

# Multiple parameters
plot_cdf(y ~ group, col = c("red", "green", "blue"), lwd = c(1, 2, 3))

# With line type and point character
plot_cdf(y ~ group, col = c("red", "green", "blue"), lty = c(1, 2, 3), lwd = 2)

# Using data frame
df <- data.frame(value = rnorm(100), group = rep(c("A", "B"), 50))
plot_cdf(value ~ group, data = df)
plot_cdf(value ~ group, data = df, col = c("red", "blue"))

# Compare two vectors
y1 <- rnorm(50)
y2 <- rnorm(50, mean = 1)
plot_cdf(y1, y2)

# Formula syntax without data (variables evaluated from environment)
widgetness <- rnorm(100)
gender <- rep(c("M", "F"), 50)
plot_cdf(widgetness ~ gender)

# Using the returned object
df <- data.frame(value = c(rnorm(50, 0), rnorm(50, 1)), group = rep(c("A", "B"), each = 50))
result <- plot_cdf(value ~ group, data = df)

# Use ECDF to find P(X <= 0.5) for group A
result$ecdfs[[1]](0.5)

# Access KS test p-value
result$ks_test$p.value

# Summarize median quantile regression
summary(result$quantile_regression_50)

```

plot_density

Plot density of a variable, optionally by another variable

Description

Plots the distribution of a variable by group, simply: `plot_density(y ~ x)`

Usage

```
plot_density(
  formula,
  y2 = NULL,
  data = NULL,
  order = NULL,
  show_means = TRUE,
  ...
)
```

Arguments

formula	Two possible uses (similar to <code>t.test()</code>): <ul style="list-style-type: none"> • Single Variable (possibly by subgroup): <code>plot_density(y)</code> or <code>plot_density(y~x)</code> • Contrast Two Variables: <code>plot_density(y1, y2)</code>
y2	optional second variable when contrasting two variables <code>plot_density(y1, y2)</code>
data	An optional data frame containing the variables in the formula.
order	Controls the order in which groups appear in the plot and legend. Use <code>-1</code> to reverse the default order. Alternatively, provide a vector specifying the exact order (e.g., <code>c("B", "A", "C")</code>). If <code>NULL</code> (default), groups are ordered by their factor levels (if the grouping variable is a factor) or sorted alphabetically/numerically. Only applies when using grouped plots.
show_means	Logical. If <code>TRUE</code> (default), shows points at means.
...	Additional arguments passed to plotting functions.

Details

Plot parameters like `col`, `lwd`, `lty`, and `pch` can be specified as:

- A single value: applied to all groups
- A vector: applied to groups in order of unique group values

Value

Invisibly returns a list with the following element:

densities A named list of density objects (class `"density"`), one for each group. Each density object contains `x` (evaluation points), `y` (density estimates), `bw` (bandwidth), and other components as returned by `density`. If no grouping variable is provided, the list contains a single element named `"all"`.

The function is primarily called for its side effect of creating a plot.

Examples

```

# Basic usage with formula syntax (no grouping)
y <- rnorm(100)
plot_density(y)

# With grouping variable
group <- rep(c("A", "B", "C"), c(30, 40, 30))
plot_density(y ~ group)

# With custom colors (scalar - same for all)
plot_density(y ~ group, col = "blue")

# With custom colors (vector - different for each group)
plot_density(y ~ group, col = c("red", "green", "blue"))

# Multiple parameters
plot_density(y ~ group, col = c("red", "green", "blue"), lwd = c(1, 2, 3))

# With line type
plot_density(y ~ group, col = c("red", "green", "blue"), lty = c(1, 2, 3), lwd = 2)

# Using data frame
df <- data.frame(value = rnorm(100), group = rep(c("A", "B"), 50))
plot_density(value ~ group, data = df)
plot_density(value ~ group, data = df, col = c("red", "blue"))

# Compare two vectors
y1 <- rnorm(50)
y2 <- rnorm(50, mean = 1)
plot_density(y1, y2)

```

plot_freq

Plot frequencies for a variable (histogram without binning)

Description

Creates a frequency plot showing the frequency of every observed value, optionally by group. Most frequent values are labeled by default.

Usage

```

plot_freq(
  formula,
  y2 = NULL,
  data = NULL,
  freq = TRUE,
  order = NULL,
  col = "dodgerblue",

```

```

    lwd = 9,
    width = NULL,
    value.labels = "auto",
    ticks.max = 30,
    show.x.value = "auto",
    show.legend = TRUE,
    legend.title = NULL,
    col.text = NULL,
    ...
)

```

Arguments

formula	Two possible uses (similar to <code>t.test()</code>): <ul style="list-style-type: none"> • Single Variable (possibly by subgroup): <code>plot_freq(y)</code> or <code>plot_freq(y~x)</code> • Contrast Two Variables: <code>plot_freq(y1, y2)</code>
y2	optional second variable when contrasting two variables <code>plot_freq(y1, y2)</code>
data	An optional data frame containing the variables in the formula.
freq	Logical. If TRUE (default), displays frequencies. If FALSE, displays percentages.
order	Controls the order in which groups appear in the plot and legend. Use -1 to reverse the default order. Alternatively, provide a vector specifying the exact order (e.g., <code>c("B", "A", "C")</code>). If NULL (default), groups are ordered by their factor levels (if the grouping variable is a factor) or sorted alphabetically/numerically. Only applies when using grouped plots or comparing two variables.
col	Color for the bars.
lwd	Line width for the frequency bars. Default is 9.
width	Numeric. Width of the frequency bars. If NULL (default), width is automatically calculated based on the spacing between values.
value.labels	Controls value labeling. If numeric, shows labels for the <code>value.labels</code> highest-frequency values (including ties). Use -1 or "all" to show all labels and 0 to show none. Use "auto" to label all values when there are 30 or fewer unique values; otherwise label the single most frequent value. For backward compatibility, TRUE is treated as -1 and FALSE as 0.
ticks.max	Integer. Maximum number of unique x values to label on the x-axis. If there are more than <code>ticks.max</code> unique values, <code>pretty()</code> ticks are used instead of labeling every value.
show.x.value	Either "auto" (default), TRUE, or FALSE. If enabled, draws a small <code>x=<value></code> label just above each frequency value label. When "auto", x labels are shown only when <code>plot_freq</code> is not already labeling all x values on the x-axis.
show.legend	Logical. If TRUE (default), displays a legend when group is specified. If FALSE, no legend is shown.
legend.title	Character string. Title for the legend when group is specified. If NULL (default), no title is shown.

col.text	Color for the value labels. If not specified, uses col for non-grouped plots or group colors for grouped plots.
...	Pass on any argument accepted by plot() e.g., xlab='x-axis', main='Distribution of X'

Details

This function creates a frequency plot where each observed value is shown with its frequency. Unlike a standard histogram, there is no binning, unlike a barplot, non-observed values of the variable are shown with 0 frequency instead of skipped.

Value

Invisibly returns a data frame with values and their frequencies.

Examples

```
# Simple example
x <- c(1, 1, 2, 2, 2, 5, 5)
plot_freq(x)

# Pass on some common \code{plot()} arguments
plot_freq(x, col = "steelblue", xlab = "Value", ylab = "Frequency", ylim=c(0,7))

# Add to an existing plot
plot_freq(x, col = "dodgerblue")

# Compare two vectors
y1 <- c(1, 1, 2, 2, 2, 5, 5)
y2 <- c(1, 2, 2, 3, 3, 3)
plot_freq(y1, y2)

# Using a data frame with grouping
df <- data.frame(value = c(1, 1, 2, 2, 2, 5, 5), group = c("A", "A", "A", "B", "B", "A", "B"))
plot_freq(value ~ 1, data = df) # single variable
plot_freq(value ~ group, data = df) # with grouping

# Control group order in legend and plot
plot_freq(value ~ group, data = df, order = c("B", "A")) # B first, then A
plot_freq(value ~ group, data = df, order = -1) # Reverse default order
```

plot_gam

Plot GAM Model

Description

Plots fitted GAM values for focal predictor, keeping any other predictors in the model at a specified quantile (default: median)

Usage

```
plot_gam(
  model,
  predictor,
  quantile.others = 50,
  col = "blue4",
  bg = adjustcolor("dodgerblue", 0.2),
  plot2 = "auto",
  col2 = NULL,
  bg2 = "gray90",
  ...
)
```

Arguments

model	A GAM model object fitted using <code>mgcv::gam()</code> .
predictor	Character string specifying the name of the predictor variable to plot on the x-axis.
quantile.others	Number between 1 and 99 for quantile at which all other predictors are held constant. Default is 50 (median).
col	Color for the prediction line. Default is "blue4".
bg	Background color for the confidence band. Default is <code>adjustcolor('dodgerblue', .2)</code> .
plot2	How to plot the distribution in the lower plot. Options: 'auto' (default, auto-select based on number of unique values), 'freq' (always plot frequencies), 'density' (always plot the density) or 'none' (neither). When 'auto', plots frequencies with predictor has less than 30 unique values, density otherwise.
col2	Color for the lines/bars in the bottom distribution plot. Default is "dodgerblue"
bg2	Background color for the bottom distribution plot. Default is "gray90".
...	Additional arguments passed to <code>plot()</code> and <code>lines()</code> .

Value

Invisibly returns a list containing:

- `predictor_values`: The sequence of predictor values used
- `predicted`: The predicted values
- `se`: The standard errors
- `lower`: Lower confidence bound ($\text{predicted} - 2 \cdot \text{se}$)
- `upper`: Upper confidence bound ($\text{predicted} + 2 \cdot \text{se}$)

Examples

```
library(mgcv)
# Fit a GAM model
data(mtcars)
mtcars$cyl <- factor(mtcars$cyl) # Convert to factor before fitting GAM
model <- gam(mpg ~ s(hp) + s(wt) + cyl, data = mtcars)

# Plot effect of hp (with other variables at median)
plot_gam(model, "hp")

# Plot effect of hp (with other variables at 25th percentile)
plot_gam(model, "hp", quantile.others = 25)

# Customize plot
plot_gam(model, "hp", main = "Effect of Horsepower", col = "blue", lwd = 2)
```

plot_means

Barplot of means

Description

Plots means, with confidence intervals, and (optionally) p-values for differences of means and interactions

Usage

```
plot_means(
  formula,
  data = NULL,
  cluster = NULL,
  tests = "auto",
  quiet = FALSE,
  order = NULL,
  legend.title = NULL,
  col = NULL,
  col.text = NULL,
  values.cex = 1,
  values.align = "top",
  values.round = 1,
  pvalue.cex = 0.9,
  pvalue.col = "gray50",
  ci.level = 95,
  buffer.top = "auto",
  ...
)
```

Arguments

formula	A formula, e.g., $y \sim x1+x2$, where $x1$ & $x2$ are grouping variables (e.g., condition indicators in a 2x2 experiment). The formula can include up to three grouping variables. The plot will show contiguous bars for $x1$, and sets of bars for $x2$ and $x3$
data	Optional data frame containing variables in the formula.
cluster	Optional clustering variable when there are repeated observations per cluster e.g., <code>cluster="participant_ID"</code> . When provided, inference for reported tests is based on regressions with clustered standard errors (via <code>lm2(..., clusters=...)</code>).
tests	specifies which comparisons of means to report. Syntax involves putting column numbers in a character string, with a - to symbolize a comparison and a + symbolizing combination. For example <code>tests="1-2"</code> reports t-test comparing columns 1 and 2. <code>tests="1-2,3-4"</code> t-tests comparing columns 1 & 2 and another 3 & 4. To run an interaction use parentheses: <code>tests="(4-3)-(2-1)"</code> and can also be combined with simple tests. Main effects can be specified using '+', for example <code>(1+2)-(3+4)</code> compares all observations in the first two columns with all observations in the next two columns.
quiet	Logical. When TRUE, suppresses console messages from <code>plot_means()</code> .
order	Controls the order of $x1$ groups (bar order and colors). Use -1 to reverse the default order (e.g., if plot shows 'male' first and 'female' second, <code>order=-1</code> will flip that).
legend.title	Character string. Title for the legend. If NULL, no title is shown.
col	Color(s) for $x1$ bars. If NULL, colors are chosen automatically.
col.text	Color for confidence intervals and other non-bar annotations. If NULL, defaults to a dark gray.
values.cex	Numeric scalar controlling text size for mean value labels (and related annotations).
values.align	Where within the bars to put the mean value labels: "top", "middle", "bottom", or "none".
values.round	Non-negative integer. Number of decimal places for mean value labels.
pvalue.cex	Numeric scalar controlling p-value label size.
pvalue.col	Color for p-value brackets/labels.
ci.level	Confidence interval level for <code>ciL/ciH</code> in <code>\$means</code> . Default is 95. You can also pass a proportion (e.g., 0.95).
buffer.top	Either "auto" (default) or a numeric value. Extra vertical headroom (as a fraction of the data y-range) added above the maximum y value to make room for annotations. When "auto", uses 0.35 when an interaction p-value is shown (scenario 2) and 0.25 otherwise.
...	Additional arguments passed to <code>plot()</code> (e.g., <code>main</code> , <code>ylim</code> , <code>ylab</code>).

Details

When `tests="auto"`, the function reports a small default set of differences-of-means tests (when applicable) and, in 2x2 designs, an interaction test:

Differences in means If `cluster` is `NULL`, these are Welch two-sample t-tests computed with `t.test(..., var.equal=FALSE)`. If `cluster` is provided, these comparisons are computed from a regression using `lm2()` with clustered standard errors.

Interaction The interaction is tested using a linear regression fit with `lm2()`, even when `cluster` is `NULL`; when `cluster` is provided, the interaction test uses clustered standard errors.

The regression-based tests use heteroskedasticity-robust inference (HC3) when `cluster` is `NULL`. HC3 is a common small-sample adjustment to White-type robust standard errors and is used to reduce sensitivity to heteroskedasticity. When `cluster` is provided, `plot_means()` instead uses clustered standard errors (robust to within-cluster correlation).

In the returned `$means` table, `ciL` and `ciH` are the lower and upper bounds of a `ci.level%` confidence interval for the mean (when available). The same confidence level is used for the confidence-interval whiskers drawn in the figure.

Value

A minimal list returned invisibly with two elements:

`means` A data frame of means (and, when available, confidence intervals) aligned to the plotting grid.

`tests` A data frame of comparisons used for p-value annotation (or `NULL` if not applicable).

Examples

```
df <- data.frame(y = rnorm(100), group = rep(c("A", "B"), 50))
plot_means(y ~ group, data = df)
```

```
df2 <- data.frame(
  y = rnorm(200),
  x1 = rep(c("A", "B"), 100),
  x2 = rep(c("X", "Y"), each = 100)
)
plot_means(y ~ x1 + x2, data = df2)
```

```
df3 <- data.frame(
  y = rnorm(600),
  x1 = rep(c("control", "treatment"), times = 300),
  x2 = rep(rep(c("low", "high"), each = 150), times = 2),
  x3 = rep(c("online", "lab"), each = 300)
)
plot_means(y ~ x1 + x2 + x3, data = df3)
```

predict.lm2

Predict method for lm2 objects

Description

Predict method for `lm2` objects

Usage

```
## S3 method for class 'lm2'  
predict(object, newdata, ...)
```

Arguments

object	An object of class lm2
newdata	An optional data frame in which to look for variables with which to predict. If omitted, the original model data is used.
...	Additional arguments passed to <code>predict.lm_robust</code> , including <code>se.fit</code> and <code>interval</code> .

Value

A vector of predicted values (or a list with `fit` and `se.fit` if `se.fit = TRUE`, or a matrix with `fit`, `lwr`, `upr` if `interval` is specified)

print.desc_var	<i>Print method for desc_var objects</i>
----------------	--

Description

Print method for desc_var objects

Usage

```
## S3 method for class 'desc_var'  
print(x, ...)
```

Arguments

x	An object of class desc_var
...	Additional arguments passed to print.data.frame

Value

Invisibly returns the original object

print.lm2 *Print method for lm2 objects*

Description

Print method for lm2 objects

Usage

```
## S3 method for class 'lm2'
print(x, notes = NULL, ...)
```

Arguments

x	An object of class lm2
notes	Logical. If TRUE (default), print explanatory notes below the table. If not specified, uses the value set when lm2() was called.
...	Additional arguments (ignored)

Value

Invisibly returns the original object

print.t.test2 *Print method for t.test2 output*

Description

Print method for t.test2 output

Usage

```
## S3 method for class 't.test2'
print(x, ...)
```

Arguments

x	An object of class t.test2
...	Additional arguments passed to print

Value

Invisibly returns the input object x. Called for its side effect of printing a formatted t-test summary to the console, including means, confidence intervals, test statistics, p-values, sample sizes, and APA-formatted results.

print.table2	<i>Print method for table2 output with centered column variable name</i>
--------------	--

Description

Print method for table2 output with centered column variable name

Print method for table2 objects

Usage

```
## S3 method for class 'table2'  
print(x, ...)
```

```
## S3 method for class 'table2'  
print(x, ...)
```

Arguments

x	An object of class table2
...	Additional arguments (ignored)

Value

Invisibly returns the input object x. Called for its side effect of printing a formatted cross-tabulation table to the console. The output includes frequencies, optional relative frequencies (row, column, or overall proportions), and chi-squared test results when applicable.

Invisibly returns the original object

resize_images	<i>Resize Images</i>
---------------	----------------------

Description

Saves images to PNG with a specified width. As input it accepts (SVG, PDF, EPS, JPG, JPEG, TIF, TIFF, BMP, PNG) Saves to subdirectory '/resized' within input folder (or same directory as file if input is a single file)

Usage

```
resize_images(path, width)
```

Arguments

path	Character string. Path to a folder containing image files, or path to a single image file.
width	Numeric vector. Target width(s) in pixels for the output PNG files. Can be a single value (recycled for all files) or a vector matching the number of files found.

Details

This function:

- Searches for image files with extensions: svg, pdf, eps, jpg, jpeg, tif, tiff, bmp, png
- Creates a "resized" subfolder in the target directory if it doesn't exist
- Converts each file to PNG format at the specified width(s)
- Saves output files as: originalname_width.png in the resized subfolder

Supported input formats:

- Vector graphics: SVG, PDF, EPS (rasterized using rsvg/magick)
- Raster images: JPG, JPEG, TIF, TIFF, BMP, PNG

Value

Invisibly returns TRUE on success.

Note

Dependencies required: rsvg, magick, and tools (base R). SVG files are rasterized using `rsvg::rsvg()`, while PDF/EPS and other formats are handled by `magick::image_read()`.

Examples

```
# Create a temporary PNG file and resize it
tmp_png <- tempfile(fileext = ".png")
grDevices::png(tmp_png, width = 400, height = 300)
old_par <- graphics::par(no.readonly = TRUE)
graphics::par(mar = c(2, 2, 1, 1))
graphics::plot(1:2, 1:2, type = "n")
grDevices::dev.off()
graphics::par(old_par)
resize_images(tmp_png, width = 80)
```

scatter.gam

*Scatter Plot with GAM Smooth Line***Description**

Creates a scatter plot with a GAM (Generalized Additive Model) smooth line. Supports both `scatter.gam(x, y)` and `scatter.gam(y ~ x)`.

Usage

```
scatter.gam(
  x,
  y,
  data.dots = TRUE,
  three.dots = FALSE,
  data = NULL,
  k = NULL,
  plot.dist = NULL,
  dot.pch = 16,
  dot.col = adjustcolor("gray", 0.7),
  jitter = FALSE,
  ...
)
```

Arguments

<code>x</code>	A numeric vector of x values, or a formula of the form $y \sim x$.
<code>y</code>	A numeric vector of y values. Not used if <code>x</code> is a formula.
<code>data.dots</code>	Logical. If TRUE, displays data on scatterplot
<code>three.dots</code>	Logical. If TRUE, divides x into tertiles and puts markers on the average x & y for each
<code>data</code>	An optional data frame containing the variables x and y.
<code>k</code>	Optional integer specifying the basis dimension for the smooth term in the GAM model (passed to <code>s(x, k=k)</code>). If NULL (default), uses the default basis dimension.
<code>plot.dist</code>	Character string specifying how to plot the distribution of x underneath the scatter plot. Options: NULL (default, auto-select based on number of unique values), "none" (no distribution plot), "plot_freq" (always use <code>plot_freq()</code>), or "hist" (always use <code>hist()</code>). When NULL, uses <code>plot_freq()</code> if there are 25 or fewer unique values, otherwise uses <code>hist()</code> .
<code>dot.pch</code>	Plotting character for data points when <code>data.dots = TRUE</code> . Default is 16 (filled circle).
<code>dot.col</code>	Color for data points when <code>data.dots = TRUE</code> . Default is <code>adjustcolor('gray', 0.7)</code> (semi-transparent gray).

jitter	Logical. If TRUE, applies a small amount of jitter to data points to reduce overplotting. Default is FALSE.
...	Additional arguments passed to plot() and gam(). Common plot arguments include: <ul style="list-style-type: none"> • main: Custom title for the plot (e.g., main = "My Title") • col: Color of the GAM smooth line (e.g., col = "red") • lwd: Line width of the GAM smooth line (e.g., lwd = 2) • xlim, ylim: Axis limits (e.g., xlim = c(0, 10)) • xlab, ylab: Axis labels (e.g., xlab = "Age")

Details

This function fits a GAM model with a smooth term for x and plots the fitted smooth line. The function uses the mgcv package's gam() function.

When three.dots = TRUE, the x variable is divided into three equal-sized groups (tertiles), and the mean x and y values for each group are plotted as points. This provides a simple summary of the relationship across the range of x .

Value

Invisibly returns the fitted GAM model object.

See Also

[scatter.smooth](#) for a simpler loess-based scatter plot smoother.

Examples

```
# Generate sample data for examples
x <- rnorm(100)
y <- 2*x + rnorm(100)

# Plot GAM smooth line only
scatter.gam(x, y)

# Equivalent call using formula syntax (y ~ x)
scatter.gam(y ~ x)

# Include scatter plot with underlying data points behind the GAM line
scatter.gam(x, y, data.dots = TRUE)

# Include summary points showing mean x and y for each tertile bin
scatter.gam(x, y, three.dots = TRUE)

# Customize the plot with a custom title, line color, and line width
scatter.gam(x, y, data.dots = TRUE, col = "red", lwd = 2, main = "GAM Fit")

# Control smoothness of the GAM line by specifying the basis dimension
scatter.gam(x, y, k = 10)
```

summary.lm2	<i>Summary method for lm2 objects</i>
-------------	---------------------------------------

Description

Summary method for lm2 objects

Usage

```
## S3 method for class 'lm2'
summary(object, ...)
```

Arguments

object	An object of class lm2
...	Additional arguments passed to print.lm2

Value

Invisibly returns the original object

t.test2	<i>Enhanced alternative to t.test()</i>
---------	---

Description

The basic t-test function in R, [t.test](#), does not report the observed difference of means, does not stipulate which mean is subtracted from which (i.e., whether it computed A-B or B-A), and presents the test results on the console in a verbose unorganized paragraph of text. `t.test2` improves on all those counts, and in addition, it reports the number of observations per group and if any observations are missing it issues a warning. It returns a dataframe instead of a list. The function is also registered as `t.test2` for the `t` generic to satisfy S3 registration checks while keeping direct calls to `t.test2(...)` unchanged.

Arguments

x	The first data argument passed to t.test . This can be a numeric vector or a formula.
...	Arguments passed to t.test

Value

A data frame with class `c("t.test2", "data.frame")` containing a single row with the following columns:

mean columns One or two columns containing group means, named after the input variables (e.g., men, women) or Group 1, Group 2 for long names.

diff column For two-sample tests, the difference between means (e.g., men-women).

ci The confidence level as a string (e.g., "95 percent").

ci.L, ci.H Lower and upper bounds of the confidence interval.

t The t-statistic.

df Degrees of freedom.

p.value The p-value.

N columns Sample sizes, named `N(group1)`, `N(group2)` or `N1`, `N2`. For paired tests, a single `N` column.

correlation For paired tests only, the correlation between pairs.

Attributes store additional information including missing value counts and test type (one-sample, two-sample, paired, Welch vs. Student).

Examples

```
# Two-sample t-test
men <- rnorm(100, mean = 5, sd = 1)
women <- rnorm(100, mean = 4.8, sd = 1)
t.test2(men, women)

# Paired t-test
x <- rnorm(50, mean = 5, sd = 1)
y <- rnorm(50, mean = 5.2, sd = 1)
t.test2(x, y, paired = TRUE)

# One-sample t-test
data <- rnorm(100, mean = 5, sd = 1)
t.test2(data, mu = 0)

# Formula syntax
data <- data.frame(y = rnorm(100), group = rep(c("A", "B"), 50))
t.test2(y ~ group, data = data)
```

Description

The function `table` does not show variable names when tabulating from a dataframe, requires running another function, `prop.table`, to tabulate proportions and yet another function, `chisq.test` to test difference of proportions. `table2` does what those three functions do, producing easier to read output, and always shows variable names.

Arguments

<code>...</code>	same arguments as <code>table</code> , plus the arguments shown below
<code>prop</code>	report a table with: <ul style="list-style-type: none"> • <code>prop="all"</code>: Proportions for full table (each cell / total) • <code>prop="row"</code>: Proportions by row ('rows' also accepted) • <code>prop="col"</code>: Proportions by columns ('cols', 'column', 'columns' also accepted)
<code>digits</code>	Number of decimal values to show for proportions
<code>chi</code>	Logical. If TRUE, performs a chi-square test on frequency table, reports results in APA format
<code>correct</code>	Logical. If TRUE, applies Yates' continuity correction for 2x2 tables in the chi-square test. Default is FALSE (no correction).

Value

A list (object of class "table2") with the following components:

- `freq`: frequency table
- `prop`: proportions table
- `chisq`: chi-square test

Examples

```
# Create example data
df <- data.frame(
  group = c("A", "A", "B", "B", "A"),
  status = c("X", "Y", "X", "Y", "X")
)

# Enhanced table with variable names (2 variables)
table2(df$group, df$status)

# Enhanced table with variable names (3 variables)
df3 <- data.frame(
  x = c("A", "A", "B", "B"),
  y = c("X", "Y", "X", "Y"),
  z = c("high", "low", "high", "low")
)
table2(df3$x, df3$y, df3$z)

# Table with proportions
```

```

table2(df$group, df$status, prop = 'all') # Overall proportions
table2(df$group, df$status, prop = 'row') # Row proportions
table2(df$group, df$status, prop = 'col') # Column proportions

# Table with chi-square test
table2(df$group, df$status, chi = TRUE,prop='all')

```

text2

Enhanced alternative to text()

Description

Adds to `text()` optional background color and verbal alignment (`align='center'`)

Arguments

<code>x, y</code>	coordinates for text placement
<code>labels</code>	text to display
<code>align</code>	alignment in relation to x coordinate ('left','center','right')
<code>bg</code>	background color
<code>cex</code>	character expansion factor
<code>pad</code>	left/right padding in percentage (e.g., .03)
<code>pad_v</code>	top/bottom padding in percentage (e.g., .25)
<code>...</code>	Additional arguments passed to text .

Value

No return value, called for side effects. Adds text with an optional background rectangle to an existing plot.

Examples

```

# Create a simple plot
plot(1:10, 1:10, type = "n", main = "text2() - Alignment & Color")

# Alignment respect to x=5
text2(5, 8, "align='left' from 5", align = "left", bg = "yellow1")
text2(5, 7, "align='right' from 5", align = "right", bg = "blue", col = "white")
text2(5, 6, "align='center' from 5", align = "center", bg = "black", col = "white")
abline(v = 5, lty = 2)

# Multiple labels with different alignments
text2(c(2, 5, 8), c(5, 5, 5),
      labels = c("Left", "Center", "Right"),
      align = c("left", "center", "right"),
      bg = c("pink", "lightblue", "lightgreen"))

```

```

# Text with custom font color (passed through ...)
text2(5, 3, "Red Text", col = "red", bg = "white")

# Padding examples
plot(1:10, 1:10, type = "n", main = "Padding Examples")

# Default padding (pad=0.03, pad_v=0.25)
text2(5, 8, "Default padding", bg = "lightblue")

# More horizontal padding
text2(5, 6, "Wide padding", pad = 0.2, bg = "lightgreen")

# More vertical padding
text2(5, 4, "Tall padding", pad_v = 0.8, bg = "lightyellow")

# Both padding increased
text2(5, 2, "Extra padding", pad = 0.15, pad_v = 0.6, bg = "pink")

```

twolines

Two-Lines Test of U-Shapes

Description

Implements the two-lines test for U-shaped (or inverted U-shaped) relationships introduced by Simonsohn (2018).

Usage

```

twolines(
  f,
  graph = 1,
  link = "gaussian",
  data = NULL,
  pngfile = "",
  quiet = FALSE
)

```

Arguments

f	A formula object specifying the model (e.g., $y \sim x_1 + x_2 + x_3$). The first predictor is the one tested for a u-shaped relationship.
graph	Integer. If 1 (default), produces a plot. If 0, no plot is generated.
link	Character string specifying the link function for the GAM model. Default is "gaussian".
data	An optional data frame containing the variables in the formula. If not provided, variables are evaluated from the calling environment.

pngfile	Optional character string. If provided, saves the plot to a PNG file with the specified filename.
quiet	Logical. If TRUE, suppresses the Robin Hood details messages. Default is FALSE.

Details

Reference: Simonsohn, Uri (2018) "Two lines: A valid alternative to the invalid testing of U-shaped relationships with quadratic regressions." AMPPS, 538-555. doi:[10.1177/2515245918805755](https://doi.org/10.1177/2515245918805755)

The test beings fitting a GAM model, predicting y with a smooth of x , and optionally with covariates. It identifies the interior most extreme value of fitted y , and adjusts from the matching x -value to set the breakpoint relying on the Robin Hood procedure introduced also by Simonsohn (2018). It then estimates the (once) interrupted regression using that breakpoint, and reports the slope and significance of the average slopes at either side of it. A U-shape is significant if the slopes are of opposite sign and are both individually significant.

Value

A list containing:

- All elements from `reg2()`: `b1`, `b2`, `p1`, `p2`, `z1`, `z2`, `u.sig`, `xc`, `glm1`, `glm2`, `rob1`, `rob2`, `msg`, `yhat.smooth`
- `yobs`: Observed y values (adjusted for covariates if present)
- `y.hat`: Fitted values from GAM
- `y.ub`, `y.lb`: Upper and lower bounds for fitted values
- `y.most`: Most extreme fitted value
- `x.most`: x -value associated with most extreme fitted value
- `f`: Formula as character string
- `bx1`, `bx2`: Linear and quadratic coefficients from preliminary quadratic regression
- `minx`: Minimum x value
- `midflat`: Median of flat region
- `midz1`, `midz2`: Z-statistics at midpoint

Examples

```
# Simple example with simulated data
set.seed(123)
x <- rnorm(100)
y <- -x^2 + rnorm(100)
data <- data.frame(x = x, y = y)
result <- twolines(y ~ x, data = data)

# With covariates
z <- rnorm(100)
y <- -x^2 + 0.5*z + rnorm(100)
data <- data.frame(x = x, y = y, z = z)
result <- twolines(y ~ x + z, data = data)
```

```
# Without data argument (variables evaluated from environment)
x <- rnorm(100)
y <- -x^2 + rnorm(100)
result <- twolines(y ~ x)
```

var_labels *Get or set variable labels*

Description

Assign labels to describe variables in a data frame. Can be called on a single variable or full data, to assign or to read existing labels.

Usage

```
var_labels(x)

var_labels(x) <- value
```

Arguments

x	A vector or a data frame.
value	Character label(s) to assign.

Details

For a vector, this reads/writes a base-R "label" attribute. For a data frame, this reads/writes the "label" attribute of each column.

Value

- If x is a data frame: a named character vector with one entry per column (missing labels are returned as NA_character_). - Otherwise: a single character string (or NA_character_).

Examples

```
df <- data.frame(x = 1:3, y = 4:6)

# Set labels for all columns
var_labels(df) <- c("this is x", "this is y")
var_labels(df)

# Set a label for a single column
var_labels(df$x) <- "this is x"
var_labels(df$x)
```

Index

chisq.test, 33
clear, 2
convert_to_sql, 3

density, 17
desc_var, 4

format_pvalue, 6

interprobe, 6

list, 9
list2, 9
lm2, 10
lm_robust, 10, 11

message2, 13

plot_cdf, 14
plot_density, 16
plot_freq, 18
plot_gam, 20
plot_means, 22
predict.lm2, 24
predict.lm_robust, 25
print.desc_var, 25
print.lm2, 26, 31
print.t.test2, 26
print.table2, 27
prop.table, 33

resize_images, 27

scatter.gam, 11, 29
scatter.smooth, 30
summary.lm2, 31

t, 31
t.test, 31
t.test2, 31
table, 33

table2, 32
text, 34
text2, 34
twolines, 35

var_labels, 37
var_labels<- (var_labels), 37