Package 'evanverse'

March 29, 2026

Type Package

Title Utility Functions for Data Analysis and Visualization

Version 0.5.1

Date 2026-03-29

Author Evan Zhou [aut, cre] (ORCID: <https://orcid.org/0009-0009-4600-8175>)

Maintainer Evan Zhou <evanzhou.bio@gmail.com>

Description A utility toolkit for data analysis and visualization in R. Provides ~50 functions covering package management, color palette management, data visualization (bar, pie, density, Venn, forest plots), statistical testing (t-test, ANOVA, chi-square, correlation, power analysis), bioinformatics utilities (gene ID conversion, GMT parsing, GEO access), and custom infix operators. Implementation follows tidyverse principles (Wickham et al. (2019) <doi:10.21105/joss.01686>).

License MIT + file LICENSE

License_is_FOSS yes

License_restricts_use no

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

URL https://github.com/evanbio/evanverse, https://evanbio.github.io/evanverse/

BugReports https://github.com/evanbio/evanverse/issues

VignetteBuilder knitr

Depends R (>= 4.1)

Imports cli, tibble, tidyr, dplyr, rlang, ggplot2, jsonlite, curl, withr, ggpubr, utils, tools, stats, grDevices, pwr, biomaRt, forestploter

Suggests BiocManager, GSEABase, GEOquery, car, ggcorrplot, knitr, psych, rmarkdown, testthat (>= 3.0.0), ggvenn, ggVennDiagram, scales, R.utils, janitor, devtools, purrr, reactable

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2026-03-29 16:00:10 UTC

Contents

---

check_pkg *Check Package Installation Status*

Description

Check whether packages are installed and optionally install missing ones. Calls `inst_pkg()` for auto-installation.

Usage

```
check_pkg(
  pkg,
  source = c("CRAN", "GitHub", "Bioconductor"),
  auto_install = FALSE,
  ...
)
```

Arguments

`pkg`	Character vector. Package names or GitHub `"user/repo"`.
`source`	Character. One of `"CRAN"`, `"GitHub"`, `"Bioconductor"`.
`auto_install`	Logical. If `TRUE`, install missing packages automatically. Default: `FALSE`.
`...`	Passed to `inst_pkg()`.

Value

A tibble with columns: `package`, `name`, `installed`, `source`.

Examples

```
check_pkg("ggplot2", source = "CRAN")
check_pkg("r-lib/devtools", source = "GitHub", auto_install = FALSE)
```

comb *Number of combinations C(n, k)*

Description

Calculates the number of ways to choose k items from n distinct items (unordered). C(n, k) = n! / (k! * (n - k)!)

Usage

```
comb(n, k)
```

Arguments

n	Non-negative integer. Total number of items.
k	Non-negative integer. Number of items to choose. Must be <= n.

Value

A numeric value. Returns 0 when k > n, 1 when k = 0 or k = n.

Examples

```
comb(8, 4)   # 70
comb(5, 2)   # 10
comb(10, 0)  # 1
comb(5, 6)   # 0
```

compile_palettes *Compile JSON Palettes into a Palette List*

Description

Read JSON files under palettes_dir/, validate content, and return a structured list of palettes. Used by data-raw/palettes.R to build the package dataset via usethis::use_data().

Usage

```
compile_palettes(palettes_dir)
```

Arguments

palettes_dir	Character. Folder containing subdirs: sequential/, diverging/, qualitative/.

Value

Invisibly returns a named list with elements sequential, diverging, qualitative.

Examples

```
compile_palettes(
  palettes_dir = system.file("extdata", "palettes", package = "evanverse")
)
```

`create_palette` *Create and Save a Custom Color Palette*

Description

Save a named color palette to a JSON file for future compilation and reuse.

Usage

```
create_palette(
  name,
  type = c("sequential", "diverging", "qualitative"),
  colors,
  color_dir,
  overwrite = FALSE
)
```

Arguments

`name`	Character. Palette name (e.g., "blues").
`type`	Character. One of "sequential", "diverging", or "qualitative".
`colors`	Character vector of HEX color values (e.g., "#E64B35" or "#E64B35B2").
`color_dir`	Character. Root folder to store palettes. Use tempdir() for examples/tests.
`overwrite`	Logical. If TRUE, overwrite existing palette file. Default: FALSE.

Value

Invisibly returns a list with `path` and `info`.

Examples

```
temp_dir <- file.path(tempdir(), "palettes")
create_palette("blues", "sequential", c("#deebf7", "#9ecae1", "#3182bd"),
  color_dir = temp_dir)
create_palette("qual_vivid", "qualitative", c("#E64B35", "#4DBBD5", "#00A087"),
  color_dir = temp_dir)

# Overwrite an existing palette explicitly
create_palette("blues", "sequential", c("#c6dbef", "#6baed6", "#2171b5"),
  color_dir = temp_dir, overwrite = TRUE)

unlink(temp_dir, recursive = TRUE)
```

df2list *Convert a data frame to a named list by grouping*

Description

Groups a data frame by one column and collects values from another column into a named list.

Usage

```
df2list(data, group_col, value_col)
```

Arguments

data	A data.frame or tibble.
group_col	Character. Column name to use as list names.
value_col	Character. Column name to collect as list values.

Value

A named list where each element is a vector of values for that group.

Examples

```
df <- data.frame(
  cell_type = c("T_cells", "T_cells", "B_cells", "B_cells"),
  marker    = c("CD3D", "CD3E", "CD79A", "MS4A1")
)
df2list(df, "cell_type", "marker")
```

df2vect *Convert a data frame to a named vector*

Description

Extracts two columns from a data frame and returns a named vector, using one column as names and the other as values. The value column type is preserved as-is.

Usage

```
df2vect(data, name_col, value_col)
```

Arguments

data	A data.frame or tibble.
name_col	Character. Column to use as vector names. Must not contain NA, empty strings, or duplicate entries; all trigger an error.
value_col	Character. Column name whose values become the vector elements. The original column type is preserved.

Value

A named vector with the same type as data[[value_col]].

Examples

```
df <- data.frame(
  gene  = c("TP53", "BRCA1", "MYC"),
  score = c(0.9, 0.7, 0.5)
)
df2vect(df, "gene", "score")
```

---

download_batch *Batch download files in parallel*

Description

Downloads multiple files concurrently using curl::multi_download().

Usage

```
download_batch(
  urls,
  dest_dir,
  overwrite = FALSE,
  resume = TRUE,
  timeout = 600,
  retries = 3
)
```

Arguments

urls	Character vector. URLs to download.
dest_dir	Character. Destination directory.
overwrite	Logical. Whether to overwrite existing files. Default: FALSE.
resume	Logical. Whether to resume from an existing non-empty destination file when overwrite = TRUE. Default: TRUE.
timeout	Integer. Timeout in seconds for each download. Default: 600.
retries	Integer. Number of retry attempts after the first failure. Default: 3.

Value

Invisibly returns a character vector of destination file paths.

`download_gene_ref` *Download gene annotation reference table from Ensembl*

Description

Downloads a standardized gene annotation table for human or mouse using `biomaRt`. Includes Ensembl ID, gene symbol, Entrez ID, gene type, chromosome location, and other metadata.

Usage

```
download_gene_ref(species = c("human", "mouse"), dest = NULL)
```

Arguments

`species`	One of `"human"` or `"mouse"`. Default: `"human"`.
`dest`	Character or `NULL`. File path to save the result as `.rds`. If `NULL` (default), the result is not saved.

Value

A `data.frame` containing gene annotation.

`download_geo` *Download a GEO series*

Description

Downloads a GEO series including expression data, supplemental files, and platform annotations.

Usage

```
download_geo(gse_id, dest_dir, retries = 2, timeout = 300)
```

Arguments

`gse_id`	Character. GEO Series accession ID (e.g., "GSE12345").
`dest_dir`	Character. Destination directory for downloaded files.
`retries`	Integer. Number of retry attempts after the first failure. Default: 2.
`timeout`	Integer. Timeout in seconds per request. Default: 300.

Value

A list with:

gse_object ExpressionSet with expression data and annotations

supplemental_files Character vector of supplemental file paths

platform_info List with `platform_id` and `gpl_files`

Examples

```
## Not run:
result <- download_geo("GSE12345", dest_dir = tempdir())
expr_data   <- Biobase::exprs(result$gse_object)
sample_info <- Biobase::pData(result$gse_object)

## End(Not run)
```

`download_url` *Download a file from a URL*

Description

Downloads a file from a URL with retry and resume support.

Usage

```
download_url(
  url,
  dest,
  overwrite = FALSE,
  resume = TRUE,
  timeout = 600,
  retries = 3
)
```

Arguments

`url`	Character string. Full URL to the file to download.
`dest`	Character string. Destination file path.
`overwrite`	Logical. Whether to overwrite an existing file. Default: FALSE. If FALSE and the file exists, download is skipped. If TRUE and `resume = TRUE`, will attempt to resume from an existing non-empty destination file. If TRUE and `resume = FALSE`, the file is downloaded from scratch.
`resume`	Logical. Whether to resume from an existing non-empty destination file when `overwrite = TRUE`. Default: TRUE.
`timeout`	Integer. Download timeout in seconds. Default: 600.
`retries`	Integer. Number of retry attempts after the first failure. Default: 3.

Value

Invisibly returns the path to the downloaded file.

Examples

```
## Not run:
download_url(
  url  = "https://raw.githubusercontent.com/tidyverse/ggplot2/main/README.md",
  dest = file.path(tempdir(), "ggplot2_readme.md")
)

## End(Not run)
```

---

file_info *Get metadata for one or more files*

Description

Returns a data.frame with file metadata for the given file path(s).

Usage

```
file_info(file)
```

Arguments

file	Character vector of file paths.

Value

A data.frame with columns: `file`, `size_MB`, `modified_time`, `path`.

Examples

```
f1 <- tempfile(fileext = ".txt")
f2 <- tempfile(fileext = ".csv")
writeLines("hello", f1)
writeLines("a,b\\n1,2", f2)
file_info(c(f1, f2))
```

file_ls *List files in a directory with metadata*

Description

Returns a data.frame with file metadata for all files in a directory.

Usage

```
file_ls(dir, recursive = FALSE, pattern = NULL)
```

Arguments

dir	Character. Directory path.
recursive	Logical. Whether to search recursively. Default: FALSE.
pattern	Optional regex to filter file names (e.g., "\.R$"). Default: NULL.

Value

A data.frame with columns: file, size_MB, modified_time, path.

Examples

```
file_ls(tempdir())
file_ls(tempdir(), pattern = "\\.R$", recursive = TRUE)
```

file_tree *Print directory tree structure*

Description

Prints the directory structure of a given dir in a tree-like format. Invisibly returns the lines so the result can be captured if needed.

Usage

```
file_tree(dir = ".", max_depth = 2)
```

Arguments

dir	Character. Root directory path. Default: ".".
max_depth	Integer. Maximum recursion depth. Default: 2.

Value

Invisibly returns a character vector of tree lines.

Examples

```
file_tree()
file_tree(".", max_depth = 3)
```

`gene2ensembl` *Convert gene symbols to Ensembl IDs*

Description

Convert gene symbols to Ensembl IDs

Usage

```
gene2ensembl(x, ref = NULL, species = c("human", "mouse"))
```

Arguments

`x`	Character vector of gene symbols.
`ref`	Data frame with columns `symbol` and `ensembl_id`. If `NULL` (default), a full reference is downloaded via `download_gene_ref` — this may trigger a network request. For examples and tests, pass `toy_gene_ref()` instead.
`species`	One of `"human"` or `"mouse"`. Controls symbol case normalization before matching. Default: `"human"`.

Value

A data.frame with columns `symbol` (original input), `symbol_std` (case-normalized), and `ensembl_id`. Unmatched entries have `NA` in `ensembl_id`.

Examples

```
ref <- toy_gene_ref(species = "human")
gene2ensembl(c("tp53", "brca1", "MYC"), ref = ref, species = "human")

ref <- toy_gene_ref(species = "mouse")
gene2ensembl(c("Zbp1", "Sftpd"), ref = ref, species = "mouse")
```

gene2entrez *Convert gene symbols to Entrez IDs*

Description

Convert gene symbols to Entrez IDs

Usage

```
gene2entrez(x, ref = NULL, species = c("human", "mouse"))
```

Arguments

x	Character vector of gene symbols.
ref	Data frame with columns symbol and entrez_id. If NULL (default), a full reference is downloaded via download_gene_ref — this may trigger a network request. For examples and tests, pass toy_gene_ref() instead.
species	One of "human" or "mouse". Controls symbol case normalization before matching. Default: "human".

Value

A data.frame with columns symbol (original input), symbol_std (case-normalized), and entrez_id. Unmatched entries have NA in entrez_id.

Examples

```
ref <- toy_gene_ref(species = "human")
gene2entrez(c("tp53", "brca1", "MYC"), ref = ref, species = "human")

ref <- toy_gene_ref(species = "mouse")
gene2entrez(c("Trp53", "Zbp1"), ref = ref, species = "mouse")
```

get_palette *Get a Color Palette*

Description

Retrieve a named palette by name and type, returning a vector of HEX colors. Automatically checks for type mismatch and provides smart suggestions.

Usage

```
get_palette(name, type = NULL, n = NULL)
```

Arguments

name	Character. Name of the palette (e.g. "qual_vivid").
type	Character. One of "sequential", "diverging", "qualitative". If NULL, type is auto-detected.
n	Integer. Number of colors to return. If NULL, returns all colors. Default is NULL.

Value

Character vector of HEX color codes.

Examples

```
get_palette("qual_vivid", type = "qualitative")
get_palette("qual_softtrio", type = "qualitative", n = 2)
get_palette("seq_blues", type = "sequential", n = 3)
get_palette("div_contrast", type = "diverging")
```

---

gmt2df *Convert a GMT file to a long-format data frame*

Description

Reads a `.gmt` gene set file and returns a long-format data frame with one row per gene.

Usage

```
gmt2df(file)
```

Arguments

file	Character. Path to a `.gmt` file.

Value

A data.frame with columns: `term`, `description`, `gene`.

Examples

```
tmp <- toy_gmt()
gmt2df(tmp)
```

gmt2list *Convert a GMT file to a named list*

Description

Reads a `.gmt` gene set file and returns a named list where each element is a character vector of gene symbols.

Usage

```
gmt2list(file)
```

Arguments

`file`	Character. Path to a `.gmt` file.

Value

A named list where each element is a character vector of gene symbols.

Examples

```
tmp <- toy_gmt()
gmt2list(tmp)
```

hex2rgb *Convert HEX Colors to RGB*

Description

Convert a character vector of HEX color codes to a data.frame with columns `hex`, `r`, `g`, `b`.

Usage

```
hex2rgb(hex)
```

Arguments

`hex`	Character vector of HEX color codes (e.g. `"#FF8000"` or `"#FF8000B2"`). Both 6-digit and 8-digit (with alpha) codes are accepted. Alpha is silently ignored. The # prefix is required. No NA values allowed.

Value

A data.frame with columns `hex`, `r`, `g`, `b`.

Examples

```
hex2rgb("#FF8000")
hex2rgb(c("#FF8000", "#00FF00"))
```

`inst_pkg` *Install R Packages from Multiple Sources*

Description

Install R packages from CRAN, GitHub, Bioconductor, or local source. Respects mirror settings from `set_mirror()`.

Usage

```
inst_pkg(
  pkg = NULL,
  source = c("CRAN", "GitHub", "Bioconductor", "Local"),
  path = NULL,
  ...
)
```

Arguments

`pkg`	Character vector. Package name(s) or GitHub `"user/repo"`. Not required for `source = "Local"`.
`source`	Character. One of `"CRAN"`, `"GitHub"`, `"Bioconductor"`, `"Local"`.
`path`	Character. Path to local package file (required for `source = "Local"`).
`...`	Passed to the underlying install function.

Value

NULL invisibly.

Examples

```
## Not run:
inst_pkg("dplyr", source = "CRAN")
inst_pkg("hadley/emo", source = "GitHub")
inst_pkg("scRNAseq", source = "Bioconductor")
inst_pkg(source = "Local", path = "mypackage.tar.gz")

## End(Not run)
```

list_palettes *List Available Color Palettes*

Description

Return a data.frame of all available palette metadata, optionally filtered by type.

Usage

```
list_palettes(type = NULL, sort = TRUE)
```

Arguments

type	Palette type(s) to filter: "sequential", "diverging", "qualitative". Default NULL returns all.
sort	Whether to sort by type, n_color, name. Default: TRUE.

Value

A data.frame with columns: name, type, n_color, colors.

Examples

```
list_palettes()
list_palettes(type = "qualitative")
list_palettes(type = c("sequential", "diverging"))
```

palettes *Built-in color palettes*

Description

A compiled list of all built-in color palettes, organized by type. Generated from JSON source files in inst/extdata/palettes/ via data-raw/palettes.R.

Usage

```
palettes
```

Format

A named list with three elements:

sequential Named list of sequential palettes; each element is a character vector of HEX color codes.

diverging Named list of diverging palettes; each element is a character vector of HEX color codes.

qualitative Named list of qualitative palettes; each element is a character vector of HEX color codes.

Source

data-raw/palettes.R

Examples

```
names(palettes)
names(palettes$qualitative)
palettes$qualitative$qual_vivid
```

---

palette_gallery *Visualize All Palettes in a Gallery View*

Description

Display palettes in a paged gallery format, returning a named list of ggplot objects.

Usage

```
palette_gallery(type = NULL, max_palettes = 30, max_row = 12, verbose = TRUE)
```

Arguments

type	Palette types to include: "sequential", "diverging", "qualitative". Default NULL returns all.
max_palettes	Number of palettes per page. Default: 30.
max_row	Max colors per row. Default: 12.
verbose	Whether to print progress info. Default: TRUE.

Value

A named list of ggplot objects (one per page).

Examples

```
palette_gallery()
palette_gallery(type = "qualitative")
palette_gallery(type = c("sequential", "diverging"), max_palettes = 10)
```

perm *Number of permutations P(n, k)*

Description

Calculates the number of ordered arrangements of k items from n distinct items. P(n, k) = n! / (n - k)!

Usage

```
perm(n, k)
```

Arguments

n	Non-negative integer. Total number of items.
k	Non-negative integer. Number of items to arrange. Must be <= n.

Value

A numeric value. Returns 0 when k > n, 1 when k = 0.

Examples

```
perm(8, 4)   # 1680
perm(5, 2)   # 20
perm(10, 0)  # 1
perm(5, 6)   # 0
```

pkg_functions *List Package Functions*

Description

List exported symbols from a package's NAMESPACE, optionally filtered by a case-insensitive keyword. Results are sorted alphabetically.

Usage

```
pkg_functions(pkg, key = NULL)
```

Arguments

pkg	Character. Package name.
key	Character. Keyword to filter function names (case-insensitive). Default: NULL.

Value

Character vector of exported names.

Examples

```
pkg_functions("evanverse")
pkg_functions("evanverse", key = "plot")
```

`pkg_version` *Check Package Versions*

Description

Check installed and latest available versions of R packages across CRAN, Bioconductor, and GitHub.

Usage

```
pkg_version(pkg)
```

Arguments

`pkg`	Character vector. Package names to check.

Value

A data.frame with columns: `package`, `version` (installed version), `latest` (latest available on CRAN/Bioc; for GitHub packages this reflects the installed remote SHA, not the actual latest remote commit), `source`. GitHub info is only available for installed packages with `RemoteType == "github"` in their DESCRIPTION.

Examples

```
## Not run:
pkg_version(c("ggplot2", "dplyr"))

## End(Not run)
```

plot_bar *Bar plot with optional grouping and sorting*

Description

Create a bar chart from a data frame with optional dodge grouping, vertical/horizontal orientation, and sorting by y values.

Usage

```
plot_bar(
  data,
  x_col,
  y_col,
  horizontal = FALSE,
  sort = FALSE,
  decreasing = TRUE,
  group_col = NULL,
  sort_by = NULL
)
```

Arguments

data	A data.frame.
x_col	Character. Column name for the x-axis. Values must be unique when group_col is NULL.
y_col	Character. Column name for the y-axis.
horizontal	Logical. If TRUE, flip to horizontal layout. Default: FALSE.
sort	Logical. Whether to sort bars by y values. Default: FALSE.
decreasing	Logical. If sort = TRUE, sort in decreasing order. Default: TRUE.
group_col	Character or NULL. Column name for dodge grouping. Default: NULL.
sort_by	Character or NULL. Required when sort = TRUE and group_col is set. Must be a valid level of the group_col column; used to order x positions by that group's y values.

Value

A ggplot object.

Examples

```
df <- data.frame(
  category = c("A", "B", "C", "D"),
  value    = c(10, 25, 15, 30)
)
plot_bar(df, x_col = "category", y_col = "value",
         sort = TRUE, horizontal = TRUE)
```

plot_density *Density plot with optional grouping and faceting*

Description

Create a univariate density plot with optional fill grouping and faceting. Density curves have a fixed black border; fill is controlled by `group_col`.

Usage

```
plot_density(
  data,
  x_col,
  group_col = NULL,
  facet_col = NULL,
  alpha = 0.7,
  adjust = 1,
  palette = NULL
)
```

Arguments

`data`	A data.frame.
`x_col`	Character. Column name of the numeric variable to plot.
`group_col`	Character or `NULL`. Column name for fill grouping. Default: `NULL`.
`facet_col`	Character or `NULL`. Column name for faceting. Default: `NULL`.
`alpha`	Numeric. Fill transparency [0, 1] (0 = fully transparent, 1 = fully opaque). Default: 0.7.
`adjust`	Numeric. Bandwidth adjustment multiplier. Default: 1.
`palette`	Character vector or `NULL`. Fill colors recycled to match the number of groups. If `NULL`, uses ggplot2 default colors. Default: `NULL`.

Value

A `ggplot` object.

Examples

```
plot_density(iris, x_col = "Sepal.Length", group_col = "Species")
```

plot_forest *Publication-ready forest plot*

Description

Produces a publication-ready forest plot with UKB-standard styling. The user supplies a data frame whose first column is the row label (`item`), plus any additional display columns (e.g. `Cases/N`). The gap column and the auto-formatted `OR (95% CI)` text column are inserted automatically at `ci_column`. Numeric p-value columns declared via `p_cols` are formatted in-place.

Usage

```
plot_forest(
  data,
  est,
  lower,
  upper,
  ci_column = 2L,
  ref_line = 1,
  xlim = NULL,
  ticks_at = NULL,
  arrow_lab = c("Lower risk", "Higher risk"),
  header = NULL,
  indent = NULL,
  bold_label = NULL,
  ci_col = "black",
  ci_sizes = 0.6,
  ci_Theight = 0.2,
  ci_digits = 2L,
  ci_sep = ", ",
  p_cols = NULL,
  p_digits = 3L,
  bold_p = TRUE,
  p_threshold = 0.05,
  align = NULL,
  background = "zebra",
  bg_col = "#F0F0F0",
  border = "three_line",
  border_width = 3,
  row_height = NULL,
  col_width = NULL,
  save = FALSE,
  dest = NULL,
  save_width = 20,
  save_height = NULL,
  theme = "default"
)
```

Arguments

data	data.frame. First column must be the label column (item). Additional columns are displayed as-is (character) or formatted if named in p_cols (must be numeric). Column order is preserved.
est	Numeric vector. Point estimates (NA = no CI drawn).
lower	Numeric vector. Lower CI bounds (same length as est).
upper	Numeric vector. Upper CI bounds (same length as est).
ci_column	Integer. Column position in the final rendered table where the gap/CI graphic is placed. Must be between 2 and ncol(data) + 1 (inclusive). Default: 2L.
ref_line	Numeric. Reference line. Default: 1 (HR/OR). Use 0 for beta coefficients.
xlim	Numeric vector of length 2. X-axis limits. NULL (default) uses c(0, 2).
ticks_at	Numeric vector. Tick positions. NULL (default) = 5 evenly spaced ticks across xlim.
arrow_lab	Character vector of length 2. Directional labels. Default: c("Lower risk", "Higher risk"). NULL = none.
header	Character vector of length ncol(data) + 2. Column header labels for the final rendered table (original columns + gap_ci + OR label). NULL (default) = use column names from data plus "gap_ci" and "OR (95% CI)". Pass "" for the gap column position.
indent	Integer vector (length = nrow(data)). Indentation level of the label column: each unit adds two leading spaces. Default: all zeros.
bold_label	Logical vector (length = nrow(data)). Which rows to bold in the label column. NULL (default) = auto-derive from indent: rows where indent == 0 are bolded (parent rows), indented sub-rows are plain.
ci_col	Character scalar or vector (length = nrow(data)). CI colour(s). NA rows are skipped automatically. Default: "black".
ci_sizes	Numeric. Point size. Default: 0.6.
ci_Theight	Numeric. CI cap height. Default: 0.2.
ci_digits	Integer. Decimal places for the auto-generated OR (95% CI) column. Default: 2L.
ci_sep	Character. Separator between lower and upper CI in the label, e.g. ", " or " - ". Default: ", ".
p_cols	Character vector. Names of numeric p-value columns in data. These are formatted to p_digits decimal places with "<0.001"-style clipping. NULL = none.
p_digits	Integer. Decimal places for p-value formatting. Default: 3L.
bold_p	TRUE (bold all non-NA p below p_threshold), FALSE (no bolding), or a logical vector (per-row control). Default: TRUE.
p_threshold	Numeric. P-value threshold for bolding when bold_p = TRUE. Default: 0.05.
align	Integer vector of length ncol(data) + 2. Alignment per column: -1 left, 0 centre, 1 right. Must cover all final columns (original + gap_ci + OR label). NULL = auto (column 1 left, all others centre).

background	Character. Row background style: "zebra", "bold_label" (shade rows where bold_label = TRUE), or "none". Default: "zebra".
bg_col	Character. Shading colour for backgrounds (scalar), or a per-row vector of length nrow(data) (overrides style). Default: "#F0F0F0".
border	Character. Border style: "three_line" or "none". Default: "three_line".
border_width	Numeric. Border line width(s). Scalar = all three lines same width; length-3 vector = top-of-header, bottom-of-header, bottom-of-body. Default: 3.
row_height	NULL (auto), numeric scalar, or numeric vector (length = total gtable rows including margins). Auto sets 8 / 12 / 10 / 15 mm for top / header / data / bottom respectively.
col_width	NULL (auto), numeric scalar, or numeric vector (length = total gtable columns). Auto rounds each default width up so the adjustment is in $[5, 10)$ mm.
save	Logical. Save output to files? Default: FALSE.
dest	Character. Destination file path (extension ignored; all four formats are saved). Required when save = TRUE.
save_width	Numeric. Output width in cm. Default: 20.
save_height	Numeric or NULL. Output height in cm. NULL = nrow(data) * 0.9 + 3.
theme	Character preset ("default") or a forestploter::forest_theme object. Default: "default".

Value

A **forestploter** plot object (gtable), returned invisibly. Display with plot() or grid::grid.draw().

Examples

```
df <- data.frame(
  item      = c("Exposure vs. control", "Unadjusted", "Fully adjusted"),
  `Cases/N` = c("", "89/4521", "89/4521"),
  p_value   = c(NA_real_, 0.001, 0.006),
  check.names = FALSE
)

p <- plot_forest(
  data       = df,
  est        = c(NA, 1.52, 1.43),
  lower      = c(NA, 1.18, 1.11),
  upper      = c(NA, 1.96, 1.85),
  ci_column  = 2L,
  indent     = c(0L, 1L, 1L),
  bold_label = c(TRUE, FALSE, FALSE),
  p_cols     = "p_value",
  xlim       = c(0.5, 3.0)
)
plot(p)
```

plot_pie *Pie chart from a vector or grouped data frame*

Description

Accepts either a character/factor vector (frequency counted automatically) or a data frame with pre-computed counts. Slices with zero count are dropped. At least two groups are required.

Usage

```
plot_pie(
  data,
  group_col = NULL,
  count_col = NULL,
  label = c("none", "count", "percent", "both"),
  palette = NULL
)
```

Arguments

data	A character/factor vector, or a data.frame.
group_col	Character. Column name for group labels (data.frame only).
count_col	Character. Column name for counts (data.frame only). Values must be non-negative.
label	Label type: "none", "count", "percent", or "both". Default: "none".
palette	Character vector or NULL. Slice fill colors recycled to match the number of groups. If NULL, uses ggplot2 default colors. Default: NULL.

Value

A ggplot object.

Examples

```
# From a vector
plot_pie(c("A", "A", "B", "C", "C", "C"))

# From a data frame
df <- data.frame(group = c("X", "Y", "Z"), count = c(10, 25, 15))
plot_pie(df, group_col = "group", count_col = "count", label = "percent")
```

plot_venn *Venn diagram for 2-4 sets*

Description

Draws a Venn diagram using either ggvenn (classic) or ggVennDiagram (gradient). Both packages must be installed (they are in Suggests).

Usage

```
plot_venn(
  set1,
  set2,
  set3 = NULL,
  set4 = NULL,
  set_names = NULL,
  method = c("classic", "gradient"),
  label = c("count", "percent", "both", "none"),
  palette = NULL,
  return_sets = FALSE
)
```

Arguments

set1, set2	Required input vectors (at least two sets).
set3, set4	Optional additional sets. Default: NULL.
set_names	Character vector of set labels. If NULL, uses the variable names of the inputs. Default: NULL.
method	Drawing method: "classic" (ggvenn) or "gradient" (ggVennDiagram). Default: "classic".
label	Label type: "count", "percent", "both", or "none". Default: "count".
palette	For "classic": character vector of fill colors, recycled to match the number of sets. For "gradient": a single RColorBrewer palette name passed to scale_fill_distiller(). If NULL, defaults are used. Default: NULL.
return_sets	Logical. If TRUE, returns list(plot, sets) instead of just the plot. Default: FALSE.

Value

A ggplot object, or a named list with elements plot and sets if return_sets = TRUE.

Examples

```
if (requireNamespace("ggvenn", quietly = TRUE)) {
  set.seed(42)
  plot_venn(sample(letters, 15), sample(letters, 12), sample(letters, 10))
}
```

preview_palette *Preview a Color Palette*

Description

Visualize a palette using various plot styles.

Usage

```
preview_palette(
  name,
  type = NULL,
  n = NULL,
  plot_type = c("bar", "pie", "point", "rect", "circle"),
  title = NULL
)
```

Arguments

name	Character. Name of the palette.
type	Character. One of "sequential", "diverging", "qualitative". If NULL, auto-detected.
n	Integer. Number of colors to use. If NULL, uses all. Default: NULL.
plot_type	Character. One of "bar", "pie", "point", "rect", "circle". Default: "bar".
title	Character. Plot title. If NULL, defaults to palette name.

Value

Invisibly returns NULL. Called for plotting side effect.

Examples

```
preview_palette("seq_blues", plot_type = "bar")
preview_palette("div_fireice", plot_type = "pie")
preview_palette("qual_vivid", n = 4, plot_type = "circle")
```

quick_anova *One-way comparison with automatic method selection*

Description

Performs a one-way ANOVA, Welch ANOVA, or Kruskal-Wallis test with automatic assumption checking and optional post-hoc comparisons.

Usage

```
quick_anova(
  data,
  group_col,
  value_col,
  method = c("auto", "anova", "welch", "kruskal"),
  post_hoc = c("auto", "none", "tukey", "welch", "wilcox"),
  alpha = 0.05
)

## S3 method for class 'quick_anova_result'
plot(
  x,
  y = NULL,
  plot_type = c("boxplot", "violin", "both"),
  add_jitter = TRUE,
  point_size = 2,
  point_alpha = 0.6,
  show_p_value = TRUE,
  p_label = c("p.format", "p.signif"),
  palette = "qual_vivid",
  ...
)
```

Arguments

data	A data frame.
group_col	Character. Column name for the grouping factor (2 or more levels).
value_col	Character. Column name for the numeric response variable.
method	One of "auto" (default), "anova", "welch", or "kruskal".
post_hoc	One of "auto" (default), "none", "tukey", "welch", or "wilcox".
alpha	Numeric. Significance level. Default 0.05.
x	A quick_anova_result object from quick_anova().
y	Ignored.
plot_type	One of "boxplot" (default), "violin", "both".

add_jitter	Logical. Add jittered points? Default TRUE.
point_size	Numeric. Jitter point size. Default 2.
point_alpha	Numeric. Jitter point transparency (0-1). Default 0.6.
show_p_value	Logical. Annotate plot with omnibus p-value? Default TRUE.
p_label	One of "p.format" (default) or "p.signif" (stars).
palette	evanverse palette name. Default "qual_vivid". NULL uses ggplot2 defaults.
...	Additional arguments passed to the internal plotting backend.

Details

Auto method selection logic:

1. Normality checked per group via Shapiro-Wilk (sample-size-adaptive thresholds — see quick_ttest for details).
2. If normality passes: Levene's test decides between classical ANOVA (equal variances) and Welch ANOVA (unequal variances).
3. If normality fails: Kruskal-Wallis is used.

Post-hoc defaults when post_hoc = "auto"**:**

- "anova" → Tukey HSD
- "welch" → Pairwise Welch t-tests (BH-adjusted)
- "kruskal" → Pairwise Wilcoxon (BH-adjusted)

Value

An object of class "quick_anova_result" (invisibly) containing:

omnibus_result List with test object, p-value, and effect size

post_hoc Post-hoc table (or NULL if none)

method_used Character: "anova", "welch", or "kruskal"

descriptive_stats Per-group summary data frame

assumption_checks List with normality and variance test results (diagnostics even when method is forced)

params List of input parameters

data Cleaned data frame (for plot() method)

Use print(result) for a brief summary, summary(result) for full details, and plot(result) for a comparison plot.

See Also

aov, oneway.test, kruskal.test

Examples

```
set.seed(123)
df <- data.frame(
  group = rep(LETTERS[1:3], each = 40),
  value = rnorm(120, mean = rep(c(0, 0.5, 1.2), each = 40), sd = 1)
)
result <- quick_anova(df, group_col = "group", value_col = "value")
print(result)
summary(result)
plot(result)
```

---

quick_chisq *Categorical association test with automatic method selection*

Description

Tests the association between two categorical variables using chi-square, Fisher's exact, or McNemar's test, with automatic method selection based on expected cell frequencies when `method = "auto"`.

Usage

```
quick_chisq(
  data,
  x_col,
  y_col,
  method = c("auto", "chisq", "fisher", "mcnemar"),
  correct = NULL,
  conf_level = 0.95,
  alpha = 0.05
)

## S3 method for class 'quick_chisq_result'
plot(
  x,
  y = NULL,
  plot_type = c("bar_grouped", "bar_stacked", "heatmap"),
  show_p_value = TRUE,
  p_label = c("p.format", "p.signif"),
  palette = "qual_vivid",
  ...
)
```

Arguments

`data`	A data frame.

x_col	Character. Column name for the first categorical variable (row variable).
y_col	Character. Column name for the second categorical variable (column variable).
method	One of "auto" (default), "chisq", "fisher", "mcnemar".
correct	Logical or NULL. Apply Yates' continuity correction? NULL (default) applies it automatically for 2×2 tables with expected frequencies between 5 and 10.
conf_level	Numeric. Confidence level for Fisher's exact test interval. Default 0.95.
alpha	Numeric. Significance level for print() and summary(). Default 0.05.
x	A quick_chisq_result object from quick_chisq().
y	Ignored.
plot_type	One of "bar_grouped" (default), "bar_stacked", or "heatmap".
show_p_value	Logical. Annotate plot with p-value? Default TRUE.
p_label	One of "p.format" (numeric, default) or "p.signif" (stars).
palette	evanverse palette name. Default "qual_vivid". NULL uses ggplot2 defaults.
...	Additional arguments passed to the internal plotting backend.

Details

Auto method selection logic:

- 2×2 table with any expected frequency < 5: Fisher's exact test
- >20\
- 2x2 table with 5 <= expected frequency < 10: Yates' correction applied
- Otherwise: standard chi-square test

WARNING: "mcnemar" is ONLY for paired/matched data (e.g., before-after measurements on the same subjects). Do NOT use for independent samples — use "chisq" or "fisher" instead.

Value

An object of class "quick_chisq_result" (invisibly) containing:

test_result An htest object from the test

method_used Character: human-readable test method label

contingency_table Observed frequency table

expected_freq Matrix of expected frequencies

pearson_residuals Pearson residuals matrix; NULL for Fisher/McNemar

effect_size Cramer's V and interpretation; NULL when statistic is unavailable

descriptive_stats Data frame with counts, proportions, and percents

auto_decision List with method selection details

params List of input parameters

data Cleaned data frame used for the test (for plot() method)

Use print(result) for a one-line summary, summary(result) for full details, and plot(result) for a visualization.

See Also

chisq.test, fisher.test, quick_ttest, quick_anova

Examples

```
set.seed(123)
df <- data.frame(
  treatment = sample(c("A", "B", "C"), 100, replace = TRUE),
  response  = sample(c("Success", "Failure"), 100, replace = TRUE,
                     prob = c(0.6, 0.4))
)
result <- quick_chisq(df, x_col = "treatment", y_col = "response")
print(result)
summary(result)
plot(result)
```

---

quick_cor *Correlation analysis with heatmap visualization*

Description

Computes pairwise correlations with p-values, optional multiple-testing correction, and a publication-ready heatmap. Supports Pearson, Spearman, and Kendall methods.

Usage

```
quick_cor(
  data,
  vars = NULL,
  method = c("pearson", "spearman", "kendall"),
  use = "pairwise.complete.obs",
 p_adjust_method = c("none", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY",
    "fdr"),
  alpha = 0.05
)

## S3 method for class 'quick_cor_result'
plot(
  x,
  y = NULL,
  type = c("full", "upper", "lower"),
  show_coef = FALSE,
  show_sig = TRUE,
  hc_order = TRUE,
  hc_method = "complete",
  palette = "gradient_rd_bu",
  lab_size = 3,
```

```
  title = NULL,
  show_axis_x = TRUE,
  show_axis_y = TRUE,
  axis_x_angle = 45,
  axis_y_angle = 0,
  axis_text_size = 10,
  sig_level = c(0.001, 0.01, 0.05),
  ...
)
```

Arguments

data	A data frame.
vars	Character vector of column names to include. NULL (default) uses all numeric columns.
method	One of "pearson" (default), "spearman", "kendall".
use	Character. Missing-value handling passed to cor(). Default "pairwise.complete.obs".
p_adjust_method	P-value adjustment method passed to p.adjust. Default "none".
alpha	Numeric. Significance threshold for identifying significant pairs. Default 0.05.
x	A quick_cor_result object from quick_cor().
y	Ignored.
type	One of "full" (default), "upper", "lower".
show_coef	Logical. Show correlation coefficients? Default FALSE.
show_sig	Logical. Show significance stars? Default TRUE. Silently disabled when show_coef = TRUE.
hc_order	Logical. Reorder by hierarchical clustering? Default TRUE.
hc_method	Clustering method. Default "complete".
palette	evanverse palette name. Default "gradient_rd_bu". NULL uses a built-in Blue-White-Red scale.
lab_size	Numeric. Label size when show_coef = TRUE. Default 3.
title	Character. Plot title. Default NULL.
show_axis_x	Logical. Show x-axis labels? Default TRUE.
show_axis_y	Logical. Show y-axis labels? Default TRUE.
axis_x_angle	Numeric. X-axis label angle. Default 45.
axis_y_angle	Numeric. Y-axis label angle. Default 0.
axis_text_size	Numeric. Axis text size. Default 10.
sig_level	Numeric vector. P-value thresholds for ***, **, *. Default c(0.001, 0.01, 0.05).
...	Additional arguments passed to the internal plotting backend.

Details

P-value computation: Uses `psych::corr.test()` when available (10-100× faster for large matrices); otherwise falls back to a `stats::cor.test()` loop.

Value

An object of class `"quick_cor_result"` (invisibly) containing:

`cor_matrix` Correlation coefficient matrix

`p_matrix` Unadjusted p-value matrix

`p_adjusted` Adjusted p-value matrix; `NULL` when `p_adjust_method = "none"`

`method_used` Correlation method used

`significant_pairs` Data frame of significant pairs

`descriptive_stats` Per-variable summary data frame

`params` List of input parameters

`data` Cleaned data frame (for `plot()` method)

Use `print(result)` for a one-line summary, `summary(result)` for full details, and `plot(result)` for the correlation heatmap.

See Also

`cor`, `cor.test`, `quick_ttest`, `quick_chisq`

Examples

```
result <- quick_cor(mtcars)
print(result)
summary(result)
plot(result)

result <- quick_cor(
  mtcars,
  vars   = c("mpg", "hp", "wt", "qsec"),
  method = "spearman",
  p_adjust_method = "BH"
)
```

quick_ttest *Two-group comparison with automatic method selection*

Description

Performs a t-test or Wilcoxon test for exactly two groups, with automatic method selection based on normality when `method = "auto"`. Always uses Welch's t-test for unpaired comparisons (no equal-variance assumption).

Usage

```
quick_ttest(
  data,
  group_col,
  value_col,
  method = c("auto", "t.test", "wilcox.test"),
  paired = FALSE,
  id_col = NULL,
  alternative = c("two.sided", "less", "greater"),
  alpha = 0.05
)

## S3 method for class 'quick_ttest_result'
plot(
  x,
  y = NULL,
  plot_type = c("boxplot", "violin", "both"),
  add_jitter = TRUE,
  point_size = 2,
  point_alpha = 0.6,
  show_p_value = TRUE,
  p_label = c("p.signif", "p.format"),
  palette = "qual_vivid",
  ...
)
```

Arguments

`data`	A data frame.
`group_col`	Character. Column name for the grouping variable (exactly 2 levels).
`value_col`	Character. Column name for the numeric response variable.
`method`	One of `"auto"` (default), `"t.test"`, `"wilcox.test"`.
`paired`	Logical. Paired test? Default `FALSE`. Requires `id_col` when `TRUE`.
`id_col`	Character. Column name for the pairing identifier. Required when `paired = TRUE`.

alternative	One of "two.sided" (default), "less", "greater".
alpha	Numeric. Significance level. Default 0.05.
x	A quick_ttest_result object from quick_ttest().
y	Ignored.
plot_type	One of "boxplot" (default), "violin", "both".
add_jitter	Logical. Add jittered points? Default TRUE.
point_size	Numeric. Jitter point size. Default 2.
point_alpha	Numeric. Jitter point transparency (0-1). Default 0.6.
show_p_value	Logical. Annotate plot with p-value? Default TRUE.
p_label	One of "p.signif" (stars, default) or "p.format" (numeric).
palette	evanverse palette name. Default "qual_vivid". NULL uses ggplot2 defaults.
...	Additional arguments passed to the internal plotting backend.

Details

Auto method selection logic:

- $n \geq 100$: t-test (CLT applies; Shapiro-Wilk unreliable at large n)
- $30 \leq n < 100$: Shapiro-Wilk at $p < 0.01$ threshold
- $n < 30$: Shapiro-Wilk at $p < 0.05$ threshold

Value

An object of class "quick_ttest_result" (invisibly) containing:

test_result An htest object from the test

method_used Character: "t.test" or "wilcox.test"

descriptive_stats Per-group summary data frame

normality_tests Shapiro-Wilk results (auto mode only); NULL when method is forced

params List of input parameters

data Cleaned data frame used for the test (for plot() method)

Use print(result) for a one-line summary, summary(result) for full details, and plot(result) for a comparison plot.

See Also

t.test, wilcox.test

Examples

```
set.seed(42)
df <- data.frame(
  group = rep(c("A", "B"), each = 30),
  value = c(rnorm(30, 5), rnorm(30, 6))
)
result <- quick_ttest(df, group_col = "group", value_col = "value")
print(result)
summary(result)
plot(result)
```

`recode_column` *Recode a column in a data frame using a named vector*

Description

Maps values in a column to new values using a named vector (`dict`). Unmatched values are replaced with `default`.

Usage

```
recode_column(data, column, dict, name = NULL, default = NA)
```

Arguments

`data`	A data.frame.
`column`	Character. Column name to recode.
`dict`	Named vector. Names are original values, values are replacements.
`name`	Character or `NULL`. Name of the output column. If `NULL` (default), the original column is overwritten. Otherwise a new column is created.
`default`	Default value for unmatched entries. Default: `NA`.

Value

A data.frame with the recoded column.

Examples

```
df <- data.frame(gene = c("TP53", "BRCA1", "EGFR", "XYZ"))
dict <- c("TP53" = "Tumor suppressor", "EGFR" = "Oncogene")
recode_column(df, "gene", dict, name = "label")
recode_column(df, "gene", dict)
```

remove_palette *Remove a Saved Palette JSON*

Description

Remove a palette JSON file by name, searching across types if needed.

Usage

```
remove_palette(name, type = NULL, color_dir)
```

Arguments

name	Character. Palette name (without '.json' suffix).
type	Character. One of "sequential", "diverging", "qualitative". If NULL, searches all types.
color_dir	Character. Root folder where palettes are stored.

Value

Invisibly TRUE if removed successfully, FALSE otherwise.

Examples

```
## Not run:
remove_palette("seq_blues", color_dir = "path/to/palettes")
remove_palette("qual_vivid", type = "qualitative", color_dir = "path/to/palettes")

## End(Not run)
```

rgb2hex *Convert RGB Values to HEX Color Codes*

Description

Convert RGB values to HEX color codes. Accepts either a numeric vector of length 3 or a data.frame with columns r, g, b (symmetric with hex2rgb()).

Usage

```
rgb2hex(rgb)
```

Arguments

rgb	A numeric vector of length 3 (e.g., c(255, 128, 0)), or a data.frame with columns r, g, b. Values must be in [0, 255]. Non-integer values are rounded to the nearest integer before conversion.

Value

A character vector of HEX color codes.

Examples

```
rgb2hex(c(255, 128, 0))
rgb2hex(hex2rgb(c("#FF8000", "#00FF00")))
```

`set_mirror` *Set CRAN/Bioconductor Mirrors*

Description

Configure CRAN and/or Bioconductor mirrors for faster package installation. All package management functions (`inst_pkg()`, `update_pkg()`, etc.) will respect these settings once set.

Usage

```
set_mirror(repo = c("all", "cran", "bioc"), mirror = "tuna")
```

Arguments

`repo`	Character. One of `"all"`, `"cran"`, `"bioc"`. Default: `"all"`.
`mirror`	Character. Mirror name. Default: `"tuna"`. CRAN: `official`, `rstudio`, `tuna`, `ustc`, `aliyun`, `sjtu`, `pku`, `hku`, `westlake`, `nju`, `sustech`. Bioc: `official`, `tuna`, `ustc`, `westlake`, `nju`.

Value

Previous mirror settings (invisibly).

Examples

```
## Not run:
set_mirror()                    # all mirrors → tuna
set_mirror("cran", "westlake")  # CRAN only
set_mirror("bioc", "ustc")      # Bioc only

## End(Not run)
```

stat_n *Calculate Required Sample Size*

Description

Computes the minimum sample size needed to achieve a target statistical power for detecting a given effect size. Supports t-tests (two-sample, one-sample, paired), ANOVA, proportion tests, correlation tests, and chi-square tests.

Usage

```
stat_n(
  power = 0.8,
  effect_size,
  test = c("t_two", "t_one", "t_paired", "anova", "proportion", "correlation", "chisq"),
  alternative = c("two.sided", "less", "greater"),
  alpha = 0.05,
  k = NULL,
  df = NULL
)
```

Arguments

power	Numeric. Target statistical power. Default: 0.8.
effect_size	Numeric. Effect size appropriate for the chosen test (must be positive). Conventions: • Cohen's d for t-tests (small 0.2, medium 0.5, large 0.8) • Cohen's f for ANOVA (small 0.1, medium 0.25, large 0.4) • Cohen's h for proportion tests (small 0.2, medium 0.5, large 0.8) • Pearson r for correlation (small 0.1, medium 0.3, large 0.5) • Cohen's w for chi-square (small 0.1, medium 0.3, large 0.5)
test	Character. Test type: "t_two" (two-sample t-test, default), "t_one" (one-sample t-test), "t_paired" (paired t-test), "anova", "proportion", "correlation", or "chisq".
alternative	Character. Direction of the alternative hypothesis: "two.sided" (default), "less", or "greater". Ignored for "anova" and "chisq".
alpha	Numeric. Significance level (Type I error rate). Default: 0.05.
k	Integer ≥ 2. Number of groups. Required when test = "anova".
df	Integer ≥ 1. Degrees of freedom. Required when test = "chisq".

Value

An object of class "power_result" (invisibly) containing:

params Named list of all input parameters

n Required sample size, rounded up. Per group for "t_two" and "anova"; number of pairs for "t_paired"; total for all others.

computed "n" — distinguishes from stat_power() results where computed = "power"

interpretation Plain-text interpretation

recommendation Practical recruitment advice

Use print(result) for a brief summary, summary(result) for full details, and plot(result) to display the sample size curve.

See Also

stat_power for computing power given sample size.

Examples

```
## Not run:
# Two-sample t-test (default)
result <- stat_n(power = 0.8, effect_size = 0.5)
print(result)
summary(result)
plot(result)

# ANOVA with 4 groups, higher power target
stat_n(power = 0.9, effect_size = 0.25, test = "anova", k = 4)

# Correlation, one-sided
stat_n(power = 0.8, effect_size = 0.3, test = "correlation",
       alternative = "greater")

## End(Not run)
```

---

stat_power *Calculate Statistical Power*

Description

Computes statistical power for a planned study given sample size, effect size, and design parameters. Supports t-tests (two-sample, one-sample, paired), ANOVA, proportion tests, correlation tests, and chi-square tests.

Usage

```
stat_power(
  n,
  effect_size,
 test = c("t_two", "t_one", "t_paired", "anova", "proportion", "correlation", "chisq"),
  alternative = c("two.sided", "less", "greater"),
  alpha = 0.05,
  k = NULL,
  df = NULL
)

## S3 method for class 'power_result'
plot(x, y = NULL, plot_range = NULL, ...)
```

Arguments

n	Integer. Sample size. Interpretation depends on test: • "t_two", "anova": per group • "t_paired": number of pairs • all others: total
effect_size	Numeric. Effect size appropriate for the chosen test (must be positive). Conventions: • Cohen's d for t-tests (small 0.2, medium 0.5, large 0.8) • Cohen's f for ANOVA (small 0.1, medium 0.25, large 0.4) • Cohen's h for proportion tests (small 0.2, medium 0.5, large 0.8) • Pearson r for correlation (small 0.1, medium 0.3, large 0.5) • Cohen's w for chi-square (small 0.1, medium 0.3, large 0.5)
test	Character. Test type: "t_two" (two-sample t-test, default), "t_one" (one-sample t-test), "t_paired" (paired t-test), "anova", "proportion", "correlation", or "chisq".
alternative	Character. Direction of the alternative hypothesis: "two.sided" (default), "less", or "greater". Ignored for "anova" and "chisq".
alpha	Numeric. Significance level (Type I error rate). Default: 0.05.
k	Integer ≥ 2. Number of groups. Required when test = "anova".
df	Integer ≥ 1. Degrees of freedom. Required when test = "chisq".
x	A power_result object returned by stat_power() or stat_n().
y	Ignored.
plot_range	Numeric vector of length 2. Custom axis range for the curve. For stat_power() results: range over sample sizes. For stat_n() results: range over effect sizes. NULL uses an automatic range.
...	Additional arguments (currently unused).

Value

An object of class "power_result" (invisibly) containing:

params Named list of all input parameters

power Computed statistical power (numeric in [0, 1])

computed "power" — distinguishes from stat_n() results where computed = "n"

interpretation Plain-text interpretation of the power value

recommendation Actionable recommendation, or NULL when power is between 0.8 and 0.95

Use print(result) for a brief summary, summary(result) for full details, and plot(result) to display the power curve.

Examples

```
## Not run:
result <- stat_power(n = 30, effect_size = 0.5)
print(result)
summary(result)
plot(result)

stat_power(n = 25, effect_size = 0.25, test = "anova", k = 3)
stat_power(n = 50, effect_size = 0.3,  test = "correlation",
           alternative = "greater")

## End(Not run)
```

---

toy_gene_ref *Generate a toy gene reference table*

Description

Generates a small simulated gene reference table for use in examples and tests. Not intended for real analyses — use download_gene_ref for a complete reference.

Usage

```
toy_gene_ref(species = c("human", "mouse"), n = 20L)
```

Arguments

species	One of "human" or "mouse". Default: "human".
n	Integer. Number of genes to return. Capped at the 100 available rows. Default: 20.

Value

A data.frame with columns `symbol`, `ensembl_id`, `entrez_id`, `gene_type`, `species`, `ensembl_version`, and `download_date`.

Examples

```
toy_gene_ref()
toy_gene_ref("mouse", n = 10)
```

`toy_gmt` *Generate a toy GMT file*

Description

Writes a small GMT gene set file to a temporary path for use in examples and tests. Not intended for real analyses.

Usage

```
toy_gmt(n = 5L)
```

Arguments

`n`	Integer. Number of gene sets to write. Capped at 5 (maximum available). Default: 5.

Value

Path to the temporary `.gmt` file.

Examples

```
tmp <- toy_gmt()
gmt2df(tmp)
```

update_pkg *Update R Packages*

Description

Update R packages from CRAN, GitHub, or Bioconductor. Supports full updates, source-specific updates, or targeted package updates.

Usage

```
update_pkg(
  pkg = NULL,
  source = c("all", "CRAN", "GitHub", "Bioconductor"),
  ...
)
```

Arguments

pkg	Character vector. Package name(s) or GitHub "user/repo". If NULL, updates all installed packages for the given source.
source	Character. One of "all", "CRAN", "GitHub", "Bioconductor". Default "all" updates all CRAN + Bioconductor packages.
...	Passed to the underlying update function.

Value

NULL invisibly.

Examples

```
## Not run:
update_pkg()                                    # all CRAN + Bioc
update_pkg(source = "CRAN")                     # all CRAN only
update_pkg("ggplot2", source = "CRAN")          # specific package
update_pkg("hadley/ggplot2", source = "GitHub")

## End(Not run)
```

view *Quick interactive table viewer*

Description

Displays a data.frame as an interactive table in the Viewer pane using `reactable`.

Usage

```
view(data, n = 10)
```

Arguments

`data`	A data.frame to display.
`n`	Integer. Number of rows per page. Default: 10.

Value

A reactable widget rendered in the Viewer pane.

Examples

```
if (requireNamespace("reactable", quietly = TRUE)) {
  view(iris)
}
```

%is% *Strict identity comparison*

Description

An infix operator for strict equality. Equivalent to `base::identical()`.

Usage

```
a %is% b
```

Arguments

`a`	First object.
`b`	Second object.

Value

`TRUE` if identical, `FALSE` otherwise.

Examples

```
1:3 %is% 1:3          # TRUE
1:3 %is% c(1, 2, 3)   # FALSE (integer vs double)
```

%map% *Case-insensitive mapping returning a named vector*

Description

Like `%match%`, but returns a named character vector instead of indices. Names are canonical entries from `table`; values are the original elements from `x`. Unmatched values are dropped.

Usage

```
x %map% table
```

Arguments

`x`	Character vector of input strings.
`table`	Character vector to match against.

Value

A named character vector. Names are canonical entries from `table`; values are the original elements from `x`. Order follows `x` (not `table`). Unmatched entries are dropped.

Examples

```
c("tp53", "brca1", "egfr") %map% c("TP53", "EGFR", "MYC")
# returns: TP53 = "tp53", EGFR = "egfr"
```

%match% *Case-insensitive match returning indices*

Description

Like `base::match()`, but ignores letter case. Useful for gene ID matching.

Usage

```
x %match% table
```

Arguments

`x`	Character vector to match.
`table`	Character vector of values to match against.

Value

An integer vector of match positions. Returns NA for non-matches.

Note

Both x and table must be **non-empty** character vectors; character(0) or non-character inputs raise an error. This differs from base::match() and %nin%, which accept empty vectors. The stricter contract is intentional for gene-ID workflows where an empty query almost always signals a upstream mistake.

Examples

```
c("tp53", "BRCA1", "egfr") %match% c("TP53", "EGFR", "MYC")
# returns: 1 NA 2
```

%nin% *Not-in operator*

Description

Tests whether elements of x are **not** present in table. Equivalent to !(x %in% table). NA behaviour follows base R semantics.

Usage

```
x %nin% table
```

Arguments

x	A vector of values to test.
table	A vector of values to test against.

Value

A logical vector.

Examples

```
c("A", "B", "C") %nin% c("B", "D")   # TRUE FALSE TRUE
1:5 %nin% c(2, 4)                    # TRUE FALSE TRUE FALSE TRUE
```

`%p%` *Paste two strings with a single space*

Description

An infix operator for string concatenation with one space between `lhs` and `rhs`.

Usage

```
lhs %p% rhs
```

Arguments

`lhs`	A character vector.
`rhs`	A character vector.

Value

A character vector of concatenated strings.

Note

Both `lhs` and `rhs` must be non-`NA` character vectors; `NA` values and non-character inputs (including `NULL`) raise an error.

Examples

```
"Hello" %p% "world"
c("hello", "good") %p% c("world", "morning")
```

Index