Package 'evanverse'

March 10, 2026

Type Package

Title Utility Functions for Data Analysis and Visualization

Version 0.4.4

Date 2026-03-10

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

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

Description A comprehensive collection of utility functions for data analysis and visualization in R. The package provides 60+ functions for data manipulation, file handling, color palette management, bioinformatics workflows, statistical analysis, plotting, and package management. Features include void value handling, custom infix operators, flexible file I/O, and publication-ready visualizations with sensible defaults. Implementation follows tidyverse principles (Wickham et al. (2019) <doi:10.21105/joss.01686>) and incorporates best practices from the R community.

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, data.table, dplyr, ggplot2, jsonlite, curl, openxlsx, readxl, tictoc, fs, rlang, withr, ggpubr, magrittr, utils, tools, stats, grDevices, pwr

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

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2026-03-10 16:10:02 UTC

Contents

`bio_palette_gallery` *bio_palette_gallery(): Visualize All Palettes in a Gallery View*

Description

Display palettes from a compiled RDS in a paged gallery format.

Usage

```
bio_palette_gallery(
  palette_rds = NULL,
  type = c("sequential", "diverging", "qualitative"),
  max_palettes = 30,
  max_row = 12,
  verbose = TRUE
)
```

Arguments

palette_rds	Path to compiled RDS. Default: internal palettes.rds from inst/extdata/.
type	Palette types to include: "sequential", "diverging", "qualitative"
max_palettes	Number of palettes per page (default: 30)
max_row	Max colors per row (default: 12)
verbose	Whether to print summary/logs (default: TRUE)

Value

A named list of ggplot objects (one per page)

---

check_pkg *Check Package Installation Status*

Description

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

Usage

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

Arguments

pkg	Character vector. Package names or GitHub repos (e.g., "user/repo").
source	Character. Package source: "CRAN", "GitHub", or "Bioconductor".
auto_install	Logical. If TRUE (default), install missing packages automatically.
...	Additional arguments passed to inst_pkg().

Value

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

Examples

```
# Check if ggplot2 is installed (will install if missing):
check_pkg("ggplot2", source = "CRAN")

# Check without auto-install:
check_pkg("r-lib/devtools", source = "GitHub", auto_install = FALSE)
```

`clear_palette_cache` *Clear Palette Cache*

Description

Clear the internal palette cache and force reload on next access. This is useful if you've updated the palette RDS file and want to reload the data.

Usage

```
clear_palette_cache()
```

Value

Invisible NULL

Examples

```
## Not run:
# Clear cache (rarely needed)
clear_palette_cache()

# Cache will be automatically reloaded on next get_palette() call
get_palette("qual_vivid", type = "qualitative")

## End(Not run)
```

comb *comb: Calculate Number of Combinations C(n, k)*

Description

Calculates the total number of ways to choose k items from n distinct items (without regard to order), i.e., the number of combinations C(n, k) = n! / (k! * (n - k)!). This function is intended for moderate n and k. For very large values, consider the 'gmp' package.

Usage

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

Arguments

n	Integer. Total number of items (non-negative integer).
k	Integer. Number of items to choose (non-negative integer, must be <= n).

Value

Numeric. The combination count C(n, k) (returns Inf for very large n).

Examples

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

combine_logic *combine_logic: Combine multiple logical vectors with a logical operator*

Description

A utility function to combine two or more logical vectors using logical AND (&) or OR (|) operations. Supports NA handling and checks for consistent vector lengths.

Usage

```
combine_logic(..., op = "&", na.rm = FALSE)
```

Arguments

...	Logical vectors to combine.
op	Operator to apply: "&" (default) or "|".
na.rm	Logical. If TRUE, treats NA values as TRUE (default is FALSE).

Value

A single logical vector of the same length as inputs.

Examples

```
x <- 1:5
combine_logic(x > 2, x %% 2 == 1)            # AND by default
combine_logic(x > 2, x %% 2 == 1, op = "|")  # OR logic
combine_logic(c(TRUE, NA), c(TRUE, TRUE), na.rm = TRUE)
```

---

compile_palettes *compile_palettes(): Compile JSON palettes into RDS*

Description

Read JSON files under `palettes_dir/`, validate content, and compile into a structured RDS file.

Usage

```
compile_palettes(palettes_dir, output_rds, log = TRUE)
```

Arguments

`palettes_dir`	Character. Folder containing subdirs: sequential/, diverging/, qualitative/ (required)
`output_rds`	Character. Path to save compiled RDS file (required). Use tempdir() for examples/tests.
`log`	Logical. Whether to log compilation events. Default: TRUE

Value

Invisibly returns RDS file path (character)

Examples

```
# Compile palettes using temporary directory:
compile_palettes(
  palettes_dir = system.file("extdata", "palettes", package = "evanverse"),
  output_rds = file.path(tempdir(), "palettes.rds")
)
```

convert_gene_id *convert_gene_id(): Convert gene identifiers using a reference table*

Description

Converts between Ensembl, Symbol, and Entrez gene IDs using a reference table. Supports both character vectors and data.frame columns. Automatically loads species-specific reference data from `data/`, or downloads if unavailable.

Usage

```
convert_gene_id(
  query,
  from = "symbol",
  to = c("ensembl_id", "entrez_id"),
  species = c("human", "mouse"),
  query_col = NULL,
  ref_table = NULL,
  keep_na = FALSE,
  preview = TRUE
)
```

Arguments

`query`	Character vector or data.frame to convert.
`from`	Source ID type (e.g., "symbol", "ensembl_id", "entrez_id").
`to`	Target ID type(s). Supports multiple.
`species`	Either `"human"` or `"mouse"`. Default `"human"`.
`query_col`	If `query` is a data.frame, the column name to convert.
`ref_table`	Optional reference table.
`keep_na`	Logical. Whether to keep unmatched rows. Default: `FALSE`.
`preview`	Logical. Whether to preview output. Default: `TRUE`.

Value

A `data.frame` containing original and converted columns.

create_palette *create_palette(): Save Custom Color Palettes as JSON*

Description

Save a named color palette (sequential, diverging, or qualitative) to a JSON file. Used for palette sharing, reuse, and future compilation.

Usage

```
create_palette(
  name,
  type = c("sequential", "diverging", "qualitative"),
  colors,
  color_dir,
  log = TRUE
)
```

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 (required). Use tempdir() for examples/tests.
log	Logical. Whether to log palette creation to a temporary log file.

Value

(Invisibly) A list with path and info.

Examples

```
# Create palette in temporary directory:
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
```

```
)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`df2list` *Convert Data Frame to Named List by Grouping*

Description

Group a data frame by one column and convert to named list. Each key becomes a list name; each value column becomes vector.

Usage

```
df2list(data, key_col, value_col, verbose = TRUE)
```

Arguments

`data`	A data.frame or tibble to be grouped.
`key_col`	Character. Column name for list names.
`value_col`	Character. Column name for list values.
`verbose`	Logical. Whether to show message. Default = TRUE.

Value

A named list, where each element is a character vector of values.

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")
```

`download_batch` *download_batch(): Batch download files using multi_download (parallel with curl)*

Description

A robust batch downloader that supports concurrent downloads with flexible options. Built on top of `curl::multi_download()` for parallelism.

Usage

```
download_batch(
  urls,
  dest_dir,
  overwrite = FALSE,
  unzip = FALSE,
  workers = 4,
  verbose = TRUE,
  timeout = 600,
  resume = FALSE,
  speed_limit = NULL,
  retries = 3
)
```

Arguments

`urls`	Character vector. List of URLs to download.
`dest_dir`	Character. Destination directory (required). Use tempdir() for examples/tests.
`overwrite`	Logical. Whether to overwrite existing files. Default: FALSE.
`unzip`	Logical. Whether to unzip after download (for supported formats). Default: FALSE.
`workers`	Integer. Number of parallel workers. Default: 4.
`verbose`	Logical. Show download progress messages. Default: TRUE.
`timeout`	Integer. Timeout in seconds for each download. Default: 600.
`resume`	Logical. Whether to resume interrupted downloads. Default: FALSE.
`speed_limit`	Numeric. Bandwidth limit in bytes/sec (e.g., 500000 = 500KB/s). Default: NULL.
`retries`	Integer. Retry attempts if download fails. Default: 3.

Value

Invisibly returns a list of downloaded (and optionally unzipped) 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"),
  remove_empty_symbol = FALSE,
  remove_na_entrez = FALSE,
  save = FALSE,
  save_path = NULL
)
```

Arguments

`species`	Organism, either `"human"` or `"mouse"`. Default is `"human"`.
`remove_empty_symbol`	Logical. Remove entries with missing gene symbol. Default: `FALSE`.
`remove_na_entrez`	Logical. Remove entries with missing Entrez ID. Default: `FALSE`.
`save`	Logical. Whether to save the result as `.rds`. Default: `FALSE`.
`save_path`	File path to save (optional). If `NULL`, will use default `gene_ref_<species>_<date>.rds`.

Value

A `data.frame` containing gene annotation.

download_geo_data *Download GEO Data Resources*

Description

Downloads GEO (Gene Expression Omnibus) datasets including expression data, supplemental files, and platform annotations with error handling and logging.

Usage

```
download_geo_data(
  gse_id,
  dest_dir,
  overwrite = FALSE,
  log = TRUE,
  log_file = NULL,
  retries = 2,
  timeout = 300
)
```

Arguments

gse_id	Character. GEO Series accession ID (e.g., "GSE12345").
dest_dir	Character. Destination directory for downloaded files.
overwrite	Logical. Whether to overwrite existing files (default: FALSE).
log	Logical. Whether to create log file (default: TRUE).
log_file	Character or NULL. Log file path (auto-generated if NULL).
retries	Numeric. Number of retry attempts (default: 2).
timeout	Numeric. Timeout in seconds (default: 300).

Details

Downloads GSEMatrix files, supplemental files, and GPL annotations. Includes retry mechanism, timeout control, and logging. Requires: GEOquery, Biobase, withr, cli.

Value

A list with components:

gse_object ExpressionSet object with expression data and annotations

supplemental_files Paths to downloaded supplemental files

platform_info Platform information (platform_id, gpl_files)

meta Download metadata (timing, file counts, etc.)

References

https://www.ncbi.nlm.nih.gov/geo/

Barrett T, Wilhite SE, Ledoux P, Evangelista C, Kim IF, Tomashevsky M, Marshall KA, Phillippy KH, Sherman PM, Holko M, Yefanov A, Lee H, Zhang N, Robertson CL, Serova N, Davis S, Soboleva A. NCBI GEO: archive for functional genomics data sets–update. Nucleic Acids Res. 2013 Jan; 41(Database issue):D991-5.

Examples

```
## Not run:
# Download GEO data (requires network connection):
result <- download_geo_data("GSE12345", dest_dir = tempdir())

# Advanced usage with custom settings:
result <- download_geo_data(
  gse_id = "GSE7305",
  dest_dir = tempdir(),
  log = TRUE,
  retries = 3,
  timeout = 600
)

# Access downloaded data:
expr_data <- Biobase::exprs(result$gse_object)
sample_info <- Biobase::pData(result$gse_object)
feature_info <- Biobase::fData(result$gse_object)

## End(Not run)
```

---

download_url *download_url(): Download File from URL*

Description

Downloads files from URLs (HTTP/HTTPS/FTP/SFTP) with robust error handling, retry mechanisms, and advanced features like resume, bandwidth limiting, and auto-extraction.

Usage

```
download_url(
  url,
  dest,
  overwrite = FALSE,
  unzip = FALSE,
  verbose = TRUE,
  timeout = 600,
  headers = NULL,
  resume = FALSE,
  speed_limit = NULL,
  retries = 3
)
```

Arguments

`url`	Character string. Full URL to the file to download. Supports HTTP, HTTPS, FTP, and SFTP protocols.
`dest`	Character string. Destination file path (required). Use file.path(tempdir(), basename(url)) for examples/tests.
`overwrite`	Logical. Whether to overwrite existing files. Default: FALSE.
`unzip`	Logical. Whether to automatically extract compressed files after download. Supports .zip, .gz, .tar.gz formats. Default: FALSE.
`verbose`	Logical. Whether to show download progress and status messages. Default: TRUE.
`timeout`	Numeric. Download timeout in seconds. Default: 600 (10 minutes).
`headers`	Named list. Custom HTTP headers for the request (e.g., list(Authorization = "Bearer token")). Default: NULL.
`resume`	Logical. Whether to attempt resuming interrupted downloads if a partial file exists. Default: FALSE.
`speed_limit`	Numeric. Bandwidth limit in bytes per second (e.g., 500000 = 500KB/s). Default: NULL (no limit).
`retries`	Integer. Number of retry attempts on download failure. Default: 3.

Details

This function provides a comprehensive solution for downloading files with:

Supported Protocols: Supports HTTP/HTTPS, FTP, and SFTP protocols.

Features: Includes retry mechanism, resume support, bandwidth control, auto-extraction, progress tracking, and custom headers.

Compression Support: Supports .zip, .gz, and .tar.gz formats.

Value

Invisible character string or vector of file paths:

If unzip = FALSE Path to the downloaded file

If unzip = TRUE Vector of paths to extracted files

Dependencies

Required packages: curl, cli, R.utils (automatically checked at runtime).

Examples

```
## Not run:
# Download a CSV file from GitHub:
download_url(
  url = "https://raw.githubusercontent.com/tidyverse/ggplot2/main/README.md",
  dest = file.path(tempdir(), "ggplot2_readme.md"),
  timeout = 30
)

# Download and extract a zip file:
download_url(
  url = "https://cran.r-project.org/src/contrib/Archive/dplyr/dplyr_0.8.0.tar.gz",
  dest = file.path(tempdir(), "dplyr.tar.gz"),
  unzip = TRUE,
  timeout = 60
)

## End(Not run)


# Quick demo with a tiny file:
download_url(
  url = "https://httpbin.org/robots.txt",
  dest = file.path(tempdir(), "robots.txt"),
  timeout = 10,
  verbose = FALSE
)
```

`file_info` *file_info: Summarise file information*

Description

Given a file or folder path (or vector), returns a data.frame containing file name, size (MB), last modified time, optional line count, and path.

Usage

```
file_info(
  paths,
  recursive = FALSE,
  count_line = TRUE,
  preview = TRUE,
  filter_pattern = NULL,
  full_name = TRUE
)
```

Arguments

paths	Character vector of file paths or a folder path.
recursive	Logical. If a folder is given, whether to search recursively. Default: FALSE.
count_line	Logical. Whether to count lines in each file. Default: TRUE.
preview	Logical. Whether to show skipped/missing messages. Default: TRUE.
filter_pattern	Optional regex to filter file names (e.g., "\.R$"). Default: NULL.
full_name	Logical. Whether to return full file paths. Default: TRUE.

Value

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

Examples

```
file_info("R")
file_info(c("README.md", "DESCRIPTION"))
file_info("R", filter_pattern = "\\.R$", recursive = TRUE)
```

---

file_tree *file_tree: Print and Log Directory Tree Structure*

Description

Print the directory structure of a given path in a tree-like format using ASCII characters for maximum compatibility across different systems. Optionally, save the result to a log file for record keeping or debugging.

Usage

```
file_tree(
  path = ".",
  max_depth = 2,
  verbose = TRUE,
  log = FALSE,
  log_path = NULL,
  file_name = NULL,
  append = FALSE
)
```

Arguments

path	Character. The target root directory path to print. Default is ".".
max_depth	Integer. Maximum depth of recursion into subdirectories. Default is 2.
verbose	Logical. Whether to print the tree to console. Default is TRUE.
log	Logical. Whether to save the tree output as a log file. Default is FALSE.

log_path	Character. Directory path to save the log file if log = TRUE. Default is tempdir().
file_name	Character. Custom file name for the log file. If NULL, a name like "file_tree_YYYYMMDD_HHMMSS.l will be used.
append	Logical. If TRUE, appends to an existing file (if present). If FALSE, overwrites the file. Default is FALSE.

Value

Invisibly returns a character vector containing each line of the file tree.

Examples

```
# Basic usage with current directory:
file_tree()
file_tree(".", max_depth = 3)


# Example with temporary directory and logging:
temp_dir <- tempdir()
file_tree(temp_dir, max_depth = 2, log = TRUE, log_path = tempdir())
```

---

forest_data *Comprehensive Forest Plot Dataset*

Description

A comprehensive dataset for demonstrating advanced forest plot functionality. Contains multiple variable types (continuous, categorical, hierarchical), multi-model comparisons, sample sizes, and color grouping information.

Format

A data frame with 33 rows and 15 columns:

variable Character vector of variable names and group headers

est Numeric vector of effect estimates (Model 1 or single model)

lower Numeric vector of lower confidence limits (Model 1)

upper Numeric vector of upper confidence limits (Model 1)

pval Numeric vector of p-values (Model 1)

est_2 Numeric vector of effect estimates for Model 2 (NA for single-model rows)

lower_2 Numeric vector of lower confidence limits for Model 2

upper_2 Numeric vector of upper confidence limits for Model 2

pval_2 Numeric vector of p-values for Model 2

est_3 Numeric vector of effect estimates for Model 3

lower_3 Numeric vector of lower confidence limits for Model 3

upper_3 Numeric vector of upper confidence limits for Model 3

pval_3 Numeric vector of p-values for Model 3

n_total Numeric vector of total sample sizes

n_event Numeric vector of event counts

event_pct Numeric vector of event percentages

color_id Character vector of color group identifiers for visualization

note Character vector of notes and model descriptions

Details

This dataset demonstrates various forest plot scenarios:

- Continuous variables (Age, BMI)
- Categorical variables with subgroups (Sex, BMI category, Treatment)
- Multi-level hierarchical structures
- Multi-model comparisons (Models 1-3 for last 3 rows)
- Sample size and event information
- Color grouping for enhanced visualizations

Source

Created for testing and demonstration of plot_forest() functionality.

`get_ext` *get_ext: Extract File Extension(s)*

Description

Extract file extension(s) from a file name or path. Supports vector input and optionally preserves compound extensions (e.g., .tar.gz) when keep_all = TRUE.

Usage

```
get_ext(paths, keep_all = FALSE, include_dot = FALSE, to_lower = FALSE)
```

Arguments

`paths`	Character vector of file names or paths.
`keep_all`	Logical. If TRUE, returns full suffix after first dot in basename. If FALSE, returns only the last extension. Default is FALSE.
`include_dot`	Logical. If TRUE, includes the leading dot in result. Default is FALSE.
`to_lower`	Logical. If TRUE, converts extensions to lowercase. Default is FALSE.

Value

Character vector of extensions.

Examples

```
get_ext("data.csv")               # "csv"
get_ext("archive.tar.gz")         # "gz"
get_ext("archive.tar.gz", TRUE)   # "tar.gz"
get_ext(c("a.R", "b.txt", "c"))   # "R" "txt" ""
get_ext("data.CSV", to_lower = TRUE)  # "csv"
```

---

get_palette *Get Palette: Load Color Palette from RDS*

Description

Load a named palette from data/palettes.rds, returning a vector of HEX colors. Automatically checks for type mismatch and provides smart suggestions.

Usage

```
get_palette(
  name,
  type = c("sequential", "diverging", "qualitative"),
  n = NULL,
  palette_rds = system.file("extdata", "palettes.rds", package = "evanverse")
)
```

Arguments

name	Character. Name of the palette (e.g. "qual_vivid").
type	Character. One of "sequential", "diverging", "qualitative".
n	Integer. Number of colors to return. If NULL, returns all colors. Default is NULL.
palette_rds	Character. Path to RDS file. Default uses system file in package.

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 *gmt2df: Convert GMT File to Long-format Data Frame*

Description

Reads a .gmt gene set file and returns a long-format data frame with one row per gene, including the gene set name and optional description.

Usage

```
gmt2df(file, verbose = TRUE)
```

Arguments

file	Character. Path to a .gmt file (supports .gmt or .gmt.gz).
verbose	Logical. Whether to show progress message. Default is TRUE.

Value

A tibble with columns: term, description, and gene.

Examples

```
## Not run:
# Requires a GMT file to run:
gmt_file <- "path/to/geneset.gmt"
result <- gmt2df(gmt_file)
head(result, 10)

## End(Not run)
```

gmt2list *gmt2list: Convert GMT File to Named List*

Description

Reads a .gmt gene set file and returns a named list, where each list element is a gene set.

Usage

```
gmt2list(file, verbose = TRUE)
```

Arguments

file	Character. Path to a .gmt file.
verbose	Logical. Whether to print message. Default is TRUE.

Value

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

Examples

```
## Not run:
# Requires a GMT file to run:
gmt_file <- "path/to/geneset.gmt"
gene_sets <- gmt2list(gmt_file)
length(gene_sets)
names(gene_sets)[1:3]

## End(Not run)
```

---

hex2rgb *Convert HEX color(s) to RGB numeric components*

Description

Convert a single HEX color string or a character vector of HEX strings to RGB numeric components. The function accepts values with or without a leading #. Messaging uses `cli` if available and falls back to `message()`.

Usage

```
hex2rgb(hex)
```

Arguments

hex	Character. A HEX color string (e.g. `"#FF8000"`) or a character vector of HEX codes. No NA values allowed.

Value

If `hex` has length 1, a named numeric vector with elements `c(r, g, b)`. If `hex` has length > 1, a named list where each element is a named numeric vector for the corresponding input.

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. Automatically 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 repo (e.g., "user/repo"). Not required for `source = "local"`.
`source`	Character. Package source: "CRAN", "GitHub", "Bioconductor", "Local". Case-insensitive, first match used.
`path`	Character. Path to local package file (required when `source = "local"`).
`...`	Additional arguments passed to `install.packages`, `install_github`, or `install`.

Value

NULL (invisibly). Side effect: installs packages.

Examples

```
## Not run:
# Install from CRAN:
inst_pkg("dplyr", source = "CRAN")

# Install from GitHub:
inst_pkg("hadley/emo", source = "GitHub")

# Install from Bioconductor:
inst_pkg("scRNAseq", source = "Bioconductor")

# Install from local file:
inst_pkg(source = "Local", path = "mypackage.tar.gz")

## End(Not run)
```

list_palettes *list_palettes(): List All Color Palettes from RDS*

Description

Load and list all available color palettes compiled into an RDS file.

Usage

```
list_palettes(
  palette_rds = system.file("extdata", "palettes.rds", package = "evanverse"),
  type = c("sequential", "diverging", "qualitative"),
  sort = TRUE,
  verbose = TRUE
)
```

Arguments

palette_rds	Path to the RDS file. Default: "inst/extdata/palettes.rds".
type	Palette type(s) to filter: "sequential", "diverging", "qualitative". Default: all.
sort	Whether to sort by type, n_color, name. Default: TRUE.
verbose	Whether to print listing details to console. 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"))
```

map_column *map_column(): Map values in a column using named vector or list*

Description

Maps values in a column of a data.frame (query) to new values using a named vector or list (map), optionally creating a new column or replacing the original.

Usage

```
map_column(
  query,
  by,
  map,
  to = "mapped",
  overwrite = FALSE,
  default = "unknown",
  preview = TRUE
)
```

Arguments

query	A data.frame containing the column to be mapped.
by	A string. Column name in query to be mapped.
map	A named vector or list. Names are original values, values are mapped values.
to	A string. Name of the column to store mapped results (if overwrite = FALSE).
overwrite	Logical. Whether to replace the by column with mapped values. Default: FALSE.
default	Default value to assign if no match is found. Default: "unknown".
preview	Logical. Whether to print preview of result (default TRUE).

Value

A data.frame with a new or modified column based on the mapping (returned invisibly).

Examples

```
df <- data.frame(gene = c("TP53", "BRCA1", "EGFR", "XYZ"))
gene_map <- c("TP53" = "Tumor suppressor", "EGFR" = "Oncogene")
map_column(df, by = "gene", map = gene_map, to = "label")
```

---

palette_cache_info *Get Palette Cache Information*

Description

Display information about the current palette cache status, including whether it's initialized, when it was loaded, and how many palettes are cached.

Usage

```
palette_cache_info()
```

Value

List with cache information (invisible)

Examples

```
## Not run:
# Check cache status
palette_cache_info()

## End(Not run)
```

---

perm *Calculate Number of Permutations A(n, k)*

Description

Calculates the total number of ways to arrange k items selected from n distinct items, i.e., the number of permutations A(n, k) = n! / (n - k)!. This function is intended for moderate n and k. For very large numbers, consider supporting the 'gmp' package.

Usage

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

Arguments

n	Integer. Total number of items (non-negative integer).
k	Integer. Number of items selected for permutation (non-negative integer, must be <= n).

Value

Numeric. The permutation count A(n, k) (returns Inf for very large n).

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 filter by a case-insensitive keyword. Results are sorted alphabetically.

Usage

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

Arguments

pkg	Character. Package name.
key	Character. Optional keyword to filter function names (case-insensitive).

Value

Character vector of exported names (invisibly).

Examples

```
# List all functions in evanverse:
pkg_functions("evanverse")

# Filter by keyword:
pkg_functions("evanverse", key = "plot")
```

pkg_management *Package Management*

Description

A unified interface for R package management across CRAN, GitHub, Bioconductor, and local sources. Provides consistent installation, checking, updating, and querying capabilities.

Details

The package management functions automatically:

- Respect mirror settings configured via `set_mirror()`
- Handle dependencies for BiocManager and devtools
- Validate package names and sources

- Provide informative error messages and progress updates

Recommended workflow:

1. (Optional) Configure mirrors: `set_mirror()`
2. Install packages: `inst_pkg()`
3. Check status: `check_pkg()`
4. Update packages: `update_pkg()`
5. Query information: `pkg_version()`, `pkg_functions()`

See Also

`install.packages` for base installation, `install_github` for GitHub packages, `install` for Bioconductor packages.

`pkg_version` *Check Package Versions*

Description

Check installed and latest available versions of R packages across CRAN, Bioconductor, and GitHub. Supports case-insensitive matching.

Usage

```
pkg_version(pkg, preview = TRUE)
```

Arguments

`pkg`	Character vector. Package names to check.
`preview`	Logical. If TRUE (default), print result to console.

Value

A data.frame with columns: `package`, `version` (installed), `latest` (available), and `source`.

Examples

```
## Not run:
# Check versions of multiple packages:
pkg_version(c("ggplot2", "dplyr"))

# Check without console preview:
result <- pkg_version(c("ggplot2", "limma"), preview = FALSE)

## End(Not run)
```

plot_bar *Bar plot with optional fill grouping, sorting, and directional layout*

Description

Create a bar chart from a data frame with optional grouping (`fill`), vertical/horizontal orientation, and sorting by values.

Usage

```
plot_bar(
  data,
  x,
  y,
  fill = NULL,
  direction = c("vertical", "horizontal"),
  sort = FALSE,
  sort_by = NULL,
  sort_dir = c("asc", "desc"),
  width = 0.7,
  ...
)
```

Arguments

`data`	A data frame.
`x`	Column name for the x-axis (quoted or unquoted).
`y`	Column name for the y-axis (quoted or unquoted).
`fill`	Optional character scalar. Column name to map to fill (grouping).
`direction`	Plot direction: "vertical" or "horizontal". Default: "vertical".
`sort`	Logical. Whether to sort bars based on y values. Default: FALSE.
`sort_by`	Optional. If `fill` is set and `sort = TRUE`, choose which level of `fill` is used for sorting.
`sort_dir`	Sorting direction: "asc" or "desc". Default: "asc".
`width`	Numeric. Bar width. Default: 0.7.
`...`	Additional args passed to `ggplot2::geom_bar()`, e.g. `alpha`, `color`.

Value

A `ggplot` object.

plot_density *plot_density: Univariate Density Plot (Fill Group, Black Outline)*

Description

Create a density plot with group color as fill, and fixed black border for all curves.

Usage

```
plot_density(
  data,
  x,
  group = NULL,
  facet = NULL,
  palette = c("#1b9e77", "#d95f02", "#7570b3"),
  alpha = 0.7,
  base_size = 14,
  xlab = NULL,
  ylab = "Density",
  title = NULL,
  legend_pos = "right",
  adjust = 1,
  show_mean = FALSE,
  mean_line_color = "red",
  add_hist = FALSE,
  hist_bins = NULL,
  add_rug = FALSE,
  theme = "minimal"
)
```

Arguments

data	data.frame. Input dataset.
x	Character. Name of numeric variable to plot.
group	Character. Grouping variable for fill color. (Optional)
facet	Character. Faceting variable. (Optional)
palette	Character vector. Fill color palette, e.g. c("#FF0000","#00FF00","#0000FF"). Will be recycled as needed. Cannot be a palette name. Default: c("#1b9e77", "#d95f02", "#7570b3")
alpha	Numeric. Fill transparency. Default: 0.7.
base_size	Numeric. Theme base font size. Default: 14.
xlab	Character. X-axis label. Default: NULL (uses variable name).
ylab	Character. Y-axis label. Default: "Density".
title	Character. Plot title. Default: NULL.

legend_pos	Character. Legend position. One of "right", "left", "top", "bottom", "none". Default: "right".
adjust	Numeric. Density bandwidth adjust. Default: 1.
show_mean	Logical. Whether to add mean line. Default: FALSE.
mean_line_color	Character. Mean line color. Default: "red".
add_hist	Logical. Whether to add histogram layer. Default: FALSE.
hist_bins	Integer. Number of histogram bins. Default: NULL (auto).
add_rug	Logical. Whether to add rug marks at bottom. Default: FALSE.
theme	Character. ggplot2 theme style. One of "minimal", "classic", "bw", "light", "dark". Default: "minimal".

Value

ggplot object.

---

plot_forest *Forest Plot with Advanced Customization*

Description

Create publication-ready forest plots with extensive customization for confidence intervals, themes, colors, borders, and layout. Designed for meta-analysis and comparative study visualizations.

Usage

```
plot_forest(
  data,
  est,
  lower,
  upper,
  ci_column,
  ref_line = 1,
  xlim = NULL,
  ticks_at = NULL,
  arrow_lab = NULL,
  sizes = 0.6,
  nudge_y = 0.2,
  theme_preset = "default",
  theme_custom = NULL,
  align_left = NULL,
  align_center = NULL,
  align_right = NULL,
  bold_group = NULL,
  bold_group_col = 1,
```

```
  bold_pvalue_cols = NULL,
  p_threshold = 0.05,
  bold_custom = NULL,
  background_style = "none",
  background_group_rows = NULL,
  background_colors = NULL,
  ci_colors = NULL,
  ci_group_ids = NULL,
  add_borders = TRUE,
  border_width = 3,
  group_headers = NULL,
  group_border_width = 6,
  custom_borders = NULL,
  height_top = 8,
  height_header = 12,
  height_main = 10,
  height_bottom = 8,
  width_left = 10,
  width_right = 10,
  width_adjust = 5,
  height_custom = NULL,
  width_custom = NULL,
  layout_verbose = TRUE,
  save_plot = FALSE,
  filename = "forest_plot",
  save_path = ".",
  save_formats = c("png", "pdf"),
  save_width = 35,
  save_height = 42,
  save_units = "cm",
  save_dpi = 300,
  save_bg = "white",
  save_overwrite = TRUE,
  save_verbose = TRUE
)
```

Arguments

`data`	Data frame containing the plot data (both text and numeric columns).
`est`	List of numeric vectors containing effect estimates. Use `list()` even for single group. Example: `list(data$estimate)` or `list(data$estimate_1, data$estimate_2)`.
`lower`	List of numeric vectors containing lower CI bounds.
`upper`	List of numeric vectors containing upper CI bounds.
`ci_column`	Integer vector specifying which column(s) to draw CI graphics. Example: `c(3)` for single group, `c(3, 7)` for two groups.
`ref_line`	Numeric. Reference line position (e.g., 1 for OR, 0 for mean difference). Default: 1.

`xlim` Numeric vector of length 2. X-axis limits. Default: `NULL` (auto-calculate).

`ticks_at` Numeric vector. X-axis tick positions. Default: `NULL` (auto-calculate).

`arrow_lab` Character vector of length 2. Labels for left and right arrows. Example: `c("Favors A", "Favors B")`. Default: `NULL`.

`sizes` Numeric. Size of CI center points. Can be:

- Single value: Automatically applied to all rows (e.g., `0.6`)
- Vector: Must match the number of data rows. If length is insufficient, later rows will have no CI displayed. To repeat a pattern, use `rep(c(0.5, 0.4, 0.3), length.out = nrow(data))`.

Default: 0.6.

`nudge_y` Numeric. Vertical nudge for multi-group CI positioning. Default: 0.2.

`theme_preset` Character. Theme preset name. Default: `"default"`. See `.get_forest_theme()` for available presets.

`theme_custom` List. Custom theme parameters to override preset. Default: `NULL`.

`align_left` Integer vector. Columns to left-align. Default: `NULL`.

`align_center` Integer vector. Columns to center-align. Default: `NULL`.

`align_right` Integer vector. Columns to right-align. Default: `NULL`.

`bold_group` Character vector. Group names to bold. Default: `NULL`.

`bold_group_col` Integer. Column containing group names. Default: 1.

`bold_pvalue_cols` Integer vector. P-value columns to bold if significant. Default: `NULL`.

`p_threshold` Numeric. P-value threshold for bolding. Default: 0.05.

`bold_custom` List of custom bold specifications. Default: `NULL`.

`background_style` Character. Background style: `"none"`, `"zebra"`, `"group"`, `"block"`. Default: `"none"`.

`background_group_rows` Integer vector. Row indices of group headers (for `"group"` and `"block"` styles). Default: `NULL`.

`background_colors` Named list of colors (primary, secondary, alternate). Default: `NULL`.

`ci_colors` Color specification for CI boxes. Can be single color, vector, or named list with mapping. Default: `NULL`.

`ci_group_ids` Optional vector of group IDs for color mapping. Default: `NULL`.

`add_borders` Logical. Add simple borders. Default: `TRUE`.

`border_width` Numeric. Border line width (mm). Default: 3.

`group_headers` List of group header specifications for multi-group plots. Default: `NULL`.

`group_border_width` Numeric. Border width for group mode (mm). Default: 6.

`custom_borders` List of custom border specifications. Default: `NULL`.

`height_top` Numeric. Top margin height (mm). Default: 8.

height_header	Numeric. Header row height (mm). Default: 12.
height_main	Numeric. Data row height (mm). Default: 10.
height_bottom	Numeric. Bottom margin height (mm). Default: 8.
width_left	Numeric. Left margin width (mm). Default: 10.
width_right	Numeric. Right margin width (mm). Default: 10.
width_adjust	Numeric. Width adjustment for data columns (mm). Default: 5.
height_custom	Named list for manual height override. Default: NULL.
width_custom	Named list for manual width override. Default: NULL.
layout_verbose	Logical. Print layout adjustment info. Default: TRUE.
save_plot	Logical. Save plot to file(s). Default: FALSE.
filename	Character. Base filename (without extension). Default: "forest_plot".
save_path	Character. Directory path for saving. Default: ".".
save_formats	Character vector. File formats to save. Default: c("png", "pdf").
save_width	Numeric. Plot width. Default: 35.
save_height	Numeric. Plot height. Default: 42.
save_units	Character. Units for width/height. Default: "cm".
save_dpi	Numeric. Resolution for raster formats. Default: 300.
save_bg	Character. Background color. Default: "white".
save_overwrite	Logical. Overwrite existing files. Default: TRUE.
save_verbose	Logical. Print save messages. Default: TRUE.

Details

Core Workflow:

The function creates a base forest plot using the **forestploter** package, then applies a series of customizations:

1. Theme application (presets or custom)
2. Text alignment adjustments
3. Bold formatting for groups and significant p-values
4. Background colors (zebra, group, or block patterns)
5. CI box colors (single, vector, or mapped)
6. Border additions (simple, group, or custom)
7. Layout adjustments (widths and heights)
8. Optional file saving

CI Color Mapping:

The ci_colors parameter accepts three formats:

- **Single color**: Applied to all rows (e.g., "#E64B35")
- **Color vector**: Must match the number of rows

- **Named list with mapping**: Maps group IDs to colors

Example of named mapping:

```
ci_colors = list(
  mapping = c("Q1" = "#DC0000", "Q2" = "#8491B4", "Q3" = "#F39B7F"),
  default = "#999999"
)
```

Value

A forest plot object (gtable). Can be displayed with `print()` or `grid.draw()`. The object contains the complete plot structure and can be saved using the built-in save functionality or standard ggplot2 methods.

Important Notes

- **Data preparation**: The `data` parameter should contain all display columns including pre-formatted text
- **CI graphics**: Use `strrep(" ", n)` to create space columns where CI graphics will be drawn
- **Layout tuning**: Use `height_custom` and `width_custom` after inspecting verbose output for fine-grained control

See Also

`forest` for the underlying plotting function, `forest_theme` for theme customization.

Examples

```
## Not run:
# Example 1: Using built-in forest_data
library(dplyr)
library(evanverse)

# Load example data
data("forest_data")

# Filter to single-model data (rows without est_2)
df <- forest_data %>%
  filter(is.na(est_2)) %>%
  filter(!is.na(est))  # Remove header rows

# Prepare display data
plot_data <- df %>%
  mutate(
    ` ` = strrep(" ", 20),
    `OR (95% CI)` = sprintf("%.2f (%.2f-%.2f)", est, lower, upper),
    `P` = ifelse(pval < 0.001, "<0.001", sprintf("%.3f", pval))
  ) %>%
  select(Variable = variable, ` `, `OR (95% CI)`, `P`)
```

```
# Create plot
p <- plot_forest(
  data = plot_data,
  est = list(df$est),
  lower = list(df$lower),
  upper = list(df$upper),
  ci_column = 2,
  ref_line = 1
)
print(p)

# Example 2: Multi-model comparison
df_multi <- forest_data %>%
  filter(!is.na(est_2))  # Multi-model rows

plot_data_multi <- df_multi %>%
  mutate(
    ` ` = strrep(" ", 20),
    `Model 1` = sprintf("%.2f (%.2f-%.2f)", est, lower, upper),
    `Model 2` = sprintf("%.2f (%.2f-%.2f)", est_2, lower_2, upper_2),
    `Model 3` = sprintf("%.2f (%.2f-%.2f)", est_3, lower_3, upper_3)
  ) %>%
  select(Variable = variable, ` `, `Model 1`, `Model 2`, `Model 3`)

p <- plot_forest(
  data = plot_data_multi,
  est = list(df_multi$est, df_multi$est_2, df_multi$est_3),
  lower = list(df_multi$lower, df_multi$lower_2, df_multi$lower_3),
  upper = list(df_multi$upper, df_multi$upper_2, df_multi$upper_3),
  ci_column = 2,
  ref_line = 1
)

# Example 3: Customized styling
p <- plot_forest(
  data = plot_data,
  est = list(df$est),
  lower = list(df$lower),
  upper = list(df$upper),
  ci_column = 2,
  xlim = c(0.5, 3),
  arrow_lab = c("Lower Risk", "Higher Risk"),
  align_left = 1,
  align_center = c(2, 3, 4),
  bold_pvalue_cols = 4,
  background_style = "zebra"
)

# Example 4: Save to files
plot_forest(
  data = plot_data,
  est = list(df$est),
  lower = list(df$lower),
```

```
  upper = list(df$upper),
  ci_column = 2,
  save_plot = TRUE,
  filename = "forest_plot",
  save_formats = c("png", "pdf")
)

## End(Not run)
```

---

plot_pie *Plot a Clean Pie Chart with Optional Inner Labels*

Description

Generate a polished pie chart from a vector or a grouped data frame. Labels (optional) are placed inside the pie slices.

Usage

```
plot_pie(
  data,
  group_col = "group",
  count_col = "count",
  label = c("none", "count", "percent", "both"),
  label_size = 4,
  label_color = "black",
  fill = c("#009076", "#C71E1D", "#15607A", "#FA8C00", "#18A1CD"),
  title = "Pie Chart",
  title_size = 14,
  title_color = "black",
  legend.position = "right",
  preview = TRUE,
  save = NULL,
  return_data = FALSE
)
```

Arguments

data	A character/factor vector or data.frame.
group_col	Group column name (for data.frame). Default: "group".
count_col	Count column name (for data.frame). Default: "count".
label	Type of label to display: "none", "count", "percent", or "both". Default: "none".
label_size	Label font size. Default: 4.
label_color	Label font color. Default: "black".
fill	Fill color vector. Default: 5-color palette.

title	Plot title. Default: "Pie Chart".
title_size	Title font size. Default: 14.
title_color	Title color. Default: "black".
legend.position	Legend position. Default: "right".
preview	Whether to print the plot. Default: TRUE.
save	Optional path to save the plot (e.g., "plot.png").
return_data	If TRUE, return list(plot = ..., data = ...). Default: FALSE.

Value

A ggplot object or list(plot, data)

---

plot_venn *Draw Venn Diagrams (2-4 sets, classic or gradient style)*

Description

A flexible and unified Venn diagram plotting function supporting both ggvenn and ggVennDiagram. Automatically handles naming, de-duplication, and visualization.

Usage

```
plot_venn(
  set1,
  set2,
  set3 = NULL,
  set4 = NULL,
  category.names = NULL,
  fill = c("skyblue", "pink", "lightgreen", "lightyellow"),
  label = "count",
  label_geom = "label",
  label_alpha = 0,
  fill_alpha = 0.5,
  label_size = 4,
  label_color = "black",
  set_color = "black",
  set_size = 5,
  edge_lty = "solid",
  edge_size = 0.8,
  title = "My Venn Diagram",
  title_size = 14,
  title_color = "#F06292",
  legend.position = "none",
  method = c("classic", "gradient"),
```

```
  digits = 1,
  label_sep = ",",
  show_outside = "auto",
  auto_scale = FALSE,
  palette = "Spectral",
  direction = 1,
  preview = TRUE,
  return_sets = FALSE,
  ...
)
```

Arguments

set1, set2, set3, set4	Input vectors. At least two sets are required.
category.names	Optional vector of set names. If NULL, variable names are used.
fill	Fill colors (for method = "classic").
label	Label type: "count", "percent", "both", or "none".
label_geom	Label geometry for ggVennDiagram: "label" or "text".
label_alpha	Background transparency for labels (only for gradient).
fill_alpha	Transparency for filled regions (only for classic).
label_size	Size of region labels.
label_color	Color of region labels.
set_color	Color of set labels and borders.
set_size	Font size for set names.
edge_lty	Line type for borders.
edge_size	Border thickness.
title	Plot title.
title_size	Title font size.
title_color	Title font color.
legend.position	Legend position. Default: "none".
method	Drawing method: "classic" (ggvenn) or "gradient" (ggVennDiagram).
digits	Decimal places for percentages (classic only).
label_sep	Separator for overlapping elements (classic only).
show_outside	Show outside elements (classic only).
auto_scale	Whether to auto-scale layout (classic only).
palette	Gradient palette name (gradient only).
direction	Palette direction (gradient only).
preview	Whether to print the plot to screen.
return_sets	If TRUE, returns a list of de-duplicated input sets.
...	Additional arguments passed to the underlying plot function.

Value

A ggplot object (and optionally a list of processed sets if `return_sets = TRUE`).

Examples

```
if (requireNamespace("ggvenn", quietly = TRUE) &&
    requireNamespace("ggVennDiagram", quietly = TRUE)) {
  set.seed(123)
  g1 <- sample(letters, 15)
  g2 <- sample(letters, 10)
  g3 <- sample(letters, 12)

  # Classic 3-set Venn
  plot_venn(g1, g2, g3, method = "classic", title = "Classic Venn")

  # Gradient 2-set Venn
  plot_venn(g1, g2, method = "gradient", title = "Gradient Venn")

  # Return sets for downstream use
  out <- plot_venn(g1, g2, return_sets = TRUE)
  names(out)
}
```

---

preview_palette *Preview Palette: Visualize a Palette from RDS*

Description

Preview the appearance of a palette from `data/palettes.rds` using various plot types. This function provides multiple visualization options to help users evaluate color palettes.

Usage

```
preview_palette(
  name,
  type = c("sequential", "diverging", "qualitative"),
  n = NULL,
  plot_type = c("bar", "pie", "point", "rect", "circle"),
  title = name,
  palette_rds = system.file("extdata", "palettes.rds", package = "evanverse"),
  preview = TRUE
)
```

Arguments

`name`	Name of the palette.
`type`	Palette type: "sequential", "diverging", "qualitative".

n	Number of colors to use (default: all).
plot_type	Plot style: "bar", "pie", "point", "rect", "circle".
title	Plot title (default: same as palette name).
palette_rds	Path to RDS file. Default: system.file("extdata", "palettes.rds", package = "evanverse").
preview	Whether to show the plot immediately. Default: TRUE.

Value

NULL (invisible), for plotting side effect.

Examples

```
# Preview sequential palette:
preview_palette("seq_blues", type = "sequential", plot_type = "bar")

# Preview diverging palette:
preview_palette("div_fireice", type = "diverging", plot_type = "pie")

# Preview qualitative palette with custom colors:
preview_palette("qual_vivid", type = "qualitative", n = 4, plot_type = "circle")
```

---

print.quick_chisq_result

Print Method for quick_chisq_result

Description

Print Method for quick_chisq_result

Usage

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

Arguments

x	An object of class quick_chisq_result
...	Additional arguments (unused)

print.quick_cor_result

Print method for quick_cor_result

Description

Print method for quick_cor_result

Usage

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

Arguments

x	A quick_cor_result object
...	Additional arguments (unused)

print.quick_ttest_result

Print method for quick_ttest_result

Description

Print method for quick_ttest_result

Usage

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

Arguments

x	A quick_ttest_result object
...	Additional arguments (unused)

quick_anova *Quick ANOVA with Automatic Method Selection*

Description

Conduct one-way ANOVA, Welch ANOVA, or Kruskal-Wallis test with automatic assumption checks, publication-ready visualization, and optional post-hoc comparisons. Designed for comparing two or more independent groups.

Usage

```
quick_anova(
  data,
  group,
  value,
  method = c("auto", "anova", "welch", "kruskal"),
  post_hoc = c("auto", "none", "tukey", "welch", "wilcox"),
  conf.level = 0.95,
  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",
  verbose = TRUE,
  ...
)
```

Arguments

data	A data frame containing the variables.
group	Column name for the grouping factor. Supports quoted or unquoted names via tidy evaluation.
value	Column name for the numeric response variable.
method	Character. One of "auto" (default), "anova", "welch", or "kruskal". When "auto", the function inspects normality and homogeneity of variances to pick an appropriate test.
post_hoc	Character. Post-hoc procedure: "auto" (default), "none", "tukey", "welch", or "wilcox". "auto" selects Tukey for ANOVA, Welch-style pairwise t-tests for Welch ANOVA, and pairwise Wilcoxon tests for Kruskal-Wallis.
conf.level	Numeric. Confidence level for the test/intervals. Default is 0.95.
plot_type	Character. One of "boxplot", "violin", or "both".
add_jitter	Logical. Add jittered points? Default TRUE.
point_size	Numeric. Size of jitter points. Default 2.

point_alpha	Numeric. Transparency for jitter points (0-1). Default 0.6.
show_p_value	Logical. Show omnibus p-value on the plot? Default TRUE.
p_label	Character. P-value display: "p.format" (default) or "p.signif" (stars).
palette	Character. Palette name from evanverse, or NULL for defaults.
verbose	Logical. Print informative messages? Default TRUE.
...	Reserved for future extensions.

Value

An object of class `quick_anova_result` with elements:

plot ggplot object of the comparison

omnibus_result List describing the main test

post_hoc Post-hoc comparison table (if requested)

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

descriptive_stats Summary statistics by group

assumption_checks Results of normality/variance checks

auto_decision Details explaining automatic selections

timestamp POSIXct timestamp of the analysis

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)
)
res <- quick_anova(df, group, value)
res$plot
summary(res)
```

---

quick_chisq *Quick Chi-Square Test with Automatic Visualization*

Description

Perform chi-square test of independence or Fisher's exact test (automatically selected based on expected frequencies) with publication-ready visualization. Designed for analyzing the association between two categorical variables.

Usage

```
quick_chisq(
  data,
  var1,
  var2,
  method = c("auto", "chisq", "fisher", "mcnemar"),
  correct = NULL,
  conf.level = 0.95,
  plot_type = c("bar_grouped", "bar_stacked", "heatmap"),
  show_p_value = TRUE,
  p_label = c("p.format", "p.signif"),
  palette = "qual_vivid",
  verbose = TRUE,
  ...
)
```

Arguments

data	A data frame containing the variables.
var1	Column name for the first categorical variable (row variable). Supports both quoted and unquoted names via NSE.
var2	Column name for the second categorical variable (column variable). Supports both quoted and unquoted names via NSE.
method	Character. Test method: "auto" (default), "chisq", "fisher", or "mcnemar". When "auto", the function intelligently selects based on expected frequencies and table size. **WARNING**: "mcnemar" is ONLY for paired/matched data (e.g., before-after measurements on the same subjects). It tests marginal homogeneity, NOT independence. Do NOT use McNemar's test for independent samples - use "chisq" or "fisher" instead.
correct	Logical or NULL. Apply Yates' continuity correction? If NULL (default), automatically applied for 2x2 tables with expected frequencies < 10.
conf.level	Numeric. Confidence level for the interval. Default is 0.95.
plot_type	Character. Type of plot: "bar_grouped" (default), "bar_stacked", or "heatmap".
show_p_value	Logical. Display p-value on the plot? Default is TRUE.
p_label	Character. P-value label format: "p.format" (numeric p-value, default) or "p.signif" (stars).
palette	Character. Color palette name from evanverse palettes. Default is "qual_vivid". Set to NULL to use ggplot2 defaults.
verbose	Logical. Print diagnostic messages? Default is TRUE.
...	Additional arguments (currently unused, reserved for future extensions).

Details

"Quick" means easy to use, not simplified or inaccurate.

This function performs full statistical testing with proper assumption checking:

Automatic Method Selection (method = "auto"): The function uses an intelligent algorithm based on expected frequencies:

- **All expected frequencies >= 5**: Standard chi-square test
- **2x2 table with any expected frequency < 5**: Fisher's exact test
- **Larger table with expected frequency < 5**: Chi-square with warning
- **2x2 table with 5 <= expected frequency < 10**: Chi-square with Yates' correction

Effect Size: Cramer's V is calculated as a measure of effect size:

- Small effect: V = 0.1
- Medium effect: V = 0.3
- Large effect: V = 0.5

Pearson Residuals: Pearson residuals are calculated for each cell as (observed - expected) / sqrt(expected):

- Values > |2| indicate significant deviation from independence
- Values > |3| indicate very significant deviation

Visualization Options:

- **bar_grouped**: Grouped bar chart (default)
- **bar_stacked**: Stacked bar chart (100\
- **heatmap**: Heatmap of Pearson residuals

Value

An object of class `quick_chisq_result` containing:

plot A ggplot object with the association visualization

test_result The htest object from `chisq.test()` or `fisher.test()`

method_used Character string of the test method used

contingency_table The contingency table (counts)

expected_freq Matrix of expected frequencies

pearson_residuals Pearson residuals for each cell

effect_size Cramer's V effect size measure

descriptive_stats Data frame with frequencies and proportions

auto_decision Details about automatic method selection

timestamp POSIXct timestamp of analysis

Important Notes

- **Categorical variables**: Both variables must be categorical or will be coerced to factors.
- **Sample size**: Fisher's exact test may be computationally intensive for large tables.
- **Missing values**: Automatically removed with a warning.
- **Low frequencies**: Cells with expected frequency < 5 may lead to unreliable results.

See Also

chisq.test, fisher.test, quick_ttest, quick_anova

Examples

```
# Example 1: Basic usage with automatic method selection
set.seed(123)
data <- 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(data, var1 = treatment, var2 = response)
print(result)

# Example 2: 2x2 table
data_2x2 <- data.frame(
  gender = rep(c("Male", "Female"), each = 50),
  disease = sample(c("Yes", "No"), 100, replace = TRUE)
)

result <- quick_chisq(data_2x2, var1 = gender, var2 = disease)

# Example 3: Customize visualization
result <- quick_chisq(data,
                      var1 = treatment,
                      var2 = response,
                      plot_type = "bar_grouped",
                      palette = "qual_balanced")

# Example 4: Manual method selection
result <- quick_chisq(data,
                      var1 = treatment,
                      var2 = response,
                      method = "chisq",
                      correct = FALSE)

# Access components
result$plot                      # ggplot object
result$test_result               # htest object
result$contingency_table         # Contingency table
result$pearson_residuals         # Pearson residuals
summary(result)                  # Detailed summary
```

---

quick_cor *Quick Correlation Analysis with Heatmap Visualization*

Description

Perform correlation analysis with automatic p-value calculation and publication-ready heatmap visualization. Supports multiple correlation methods and significance testing with optional multiple testing correction.

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"),
  sig_level = c(0.001, 0.01, 0.05),
  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,
  verbose = TRUE,
  ...
)
```

Arguments

data	A data frame containing numeric variables.
vars	Optional character vector specifying which variables to include. If NULL (default), all numeric columns will be used.
method	Character. Correlation method: "pearson" (default), "spearman", or "kendall".
use	Character. Method for handling missing values, passed to cor(). Default is "pairwise.complete.obs". Other options: "everything", "all.obs", "complete.obs", "na.or.complete".
p_adjust_method	Character. Method for p-value adjustment for multiple testing. Default is "none". Options: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See p.adjust.
sig_level	Numeric vector. Significance levels for star annotations. Default is c(0.001, 0.01, 0.05) corresponding to ***, **, *.

type	Character. Type of heatmap: "full" (default), "upper", or "lower".
show_coef	Logical. Display correlation coefficients on the heatmap? Default is FALSE.
show_sig	Logical. Display significance stars on the heatmap? Default is TRUE.
hc_order	Logical. Reorder variables using hierarchical clustering? Default is TRUE.
hc_method	Character. Hierarchical clustering method if hc_order = TRUE. Default is "complete". See hclust.
palette	Character. Color palette name from evanverse palettes. Default is "gradient_rd_bu" (diverging Red-Blue palette, recommended for correlation matrices). Set to NULL to use ggplot2 defaults. Other diverging options: "piyg", "earthy_diverge", "fire_ice_duo".
lab_size	Numeric. Size of coefficient labels if show_coef = TRUE. Default is 3.
title	Character. Plot title. Default is NULL (no title).
show_axis_x	Logical. Display x-axis labels? Default is TRUE.
show_axis_y	Logical. Display y-axis labels? Default is TRUE.
axis_x_angle	Numeric. Rotation angle for x-axis labels in degrees. Default is 45. Common values: 0 (horizontal), 45 (diagonal), 90 (vertical).
axis_y_angle	Numeric. Rotation angle for y-axis labels in degrees. Default is 0 (horizontal).
axis_text_size	Numeric. Font size for axis labels. Default is 10.
verbose	Logical. Print diagnostic messages? Default is TRUE.
...	Additional arguments (currently unused, reserved for future extensions).

Details

"Quick" means easy to use, not simplified or inaccurate.

This function performs complete correlation analysis with proper statistical testing:

Correlation Methods:

- **Pearson**: Measures linear relationships, assumes normality
- **Spearman**: Rank-based, robust to outliers and non-normality
- **Kendall**: Rank-based, better for small samples or many ties

P-value Calculation: P-values are calculated for each pairwise correlation. The function automatically uses `psych::corr.test()` if the psych package is installed, which provides significantly faster computation (10-100x speedup for large matrices) compared to the base R `stats::cor.test()` loop. If psych is not available, the function gracefully falls back to the base R implementation.

For large correlation matrices with many tests, consider using `p_adjust_method` to control for multiple testing (e.g., "bonferroni" or "fdr").

Performance tip: Install the psych package for faster p-value computation: `install.packages("psych")`

Visualization: The heatmap includes:

- Color-coded correlation coefficients (red = positive, blue = negative)
- Optional significance stars (***, **, *)
- Optional coefficient values
- Hierarchical clustering to group similar variables
- Publication-ready styling

Value

An object of class `quick_cor_result` containing:

plot A ggplot object with the correlation heatmap

cor_matrix Correlation coefficient matrix

p_matrix P-value matrix (unadjusted)

p_adjusted Adjusted p-value matrix (if p_adjust_method != "none")

method_used Correlation method used

significant_pairs Data frame of significant correlation pairs

descriptive_stats Descriptive statistics for each variable

parameters List of analysis parameters

timestamp POSIXct timestamp of analysis

Important Notes

- **Numeric variables only**: The function automatically selects numeric columns or uses the variables specified in `vars`.
- **Constant variables**: Variables with zero variance are automatically removed with a warning.
- **Sample size**: The function will warn if sample sizes are very small (n < 5) after removing missing values.
- **Missing values**: Handled according to the `use` parameter. "pairwise.complete.obs" is recommended for optimal sample size usage.
- **Optional dependencies**: For optimal performance, install `psych` (fast p-value computation) and `ggcorrplot` (heatmap visualization). The function will work without them but may be slower or use fallback plotting.

See Also

`cor`, `cor.test`

Examples

```
if (requireNamespace("ggcorrplot", quietly = TRUE)) {
  # Example 1: Basic correlation analysis
  result <- quick_cor(mtcars)
  print(result)

  # Example 2: Spearman correlation with specific variables
  result <- quick_cor(
    mtcars,
    vars = c("mpg", "hp", "wt", "qsec"),
    method = "spearman"
  )

  # Example 3: Upper triangular with Bonferroni correction
  result <- quick_cor(
```

```
  iris,
  type = "upper",
  p_adjust_method = "bonferroni",
  show_coef = TRUE
)

# Example 4: Custom palette and title
result <- quick_cor(
  mtcars,
  palette = "gradient_rd_bu",
  title = "Correlation Matrix of mtcars Dataset",
  hc_order = TRUE
)

# Example 5: Customize axis labels
result <- quick_cor(
  mtcars,
  axis_x_angle = 90,      # Vertical x-axis labels
  axis_text_size = 12,    # Larger text
  show_axis_y = FALSE     # Hide y-axis labels
)

# Access components
result$plot                 # ggplot object
result$cor_matrix           # Correlation matrix
result$significant_pairs    # Significant pairs
summary(result)             # Detailed summary
}
```

---

quick_ttest *Quick t-test with Automatic Visualization*

Description

Perform t-test or Wilcoxon test (automatically selected based on data characteristics and sample size) with publication-ready visualization. Designed for comparing **two groups only**.

Usage

```
quick_ttest(
  data,
  group,
  value,
  method = c("auto", "t.test", "wilcox.test"),
  paired = FALSE,
  id,
  alternative = c("two.sided", "less", "greater"),
  var.equal = NULL,
```

```
  conf.level = 0.95,
  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",
  verbose = TRUE,
  ...
)
```

Arguments

data	A data frame containing the variables.
group	Column name for the grouping variable (must have exactly 2 levels). Supports both quoted and unquoted names via NSE.
value	Column name for the numeric values to compare. Supports both quoted and unquoted names via NSE.
method	Character. Test method: "auto" (default), "t.test", or "wilcox.test". When "auto", the function intelligently selects based on normality and sample size.
paired	Logical. Whether to perform a paired test. Default is FALSE. If TRUE, the id parameter must be specified to match pairs.
id	Column name for the pairing ID variable (required when paired = TRUE). Each unique ID should appear exactly once in each group. Supports both quoted and unquoted names via NSE.
alternative	Character. Alternative hypothesis: "two.sided" (default), "less", or "greater".
var.equal	Logical or NULL. Assume equal variances? If NULL (default), automatically tested using Levene's test (ignored for paired tests).
conf.level	Numeric. Confidence level for the interval. Default is 0.95.
plot_type	Character. Type of plot: "boxplot" (default), "violin", or "both".
add_jitter	Logical. Add jittered points to the plot? Default is TRUE.
point_size	Numeric. Size of the points. Default is 2.
point_alpha	Numeric. Transparency of points (0-1). Default is 0.6.
show_p_value	Logical. Display p-value on the plot? Default is TRUE.
p_label	Character. P-value label format: "p.signif" (stars, default) or "p.format" (numeric p-value).
palette	Character. Color palette name from evanverse palettes. Default is "qual_vivid". Set to NULL to use ggplot2 defaults.
verbose	Logical. Print diagnostic messages? Default is TRUE.
...	Additional arguments (currently unused, reserved for future extensions).

Details

"Quick" means easy to use, not simplified or inaccurate.

This function performs full statistical testing with proper assumption checking:

Automatic Method Selection (method = "auto"): The function uses an intelligent algorithm that considers both normality and sample size:

- **Large samples (n >= 100 per group)**: Prefers t-test due to Central Limit Theorem, even if Shapiro-Wilk rejects normality (which becomes overly sensitive in large samples).
- **Medium samples (30 <= n < 100)**: Uses Shapiro-Wilk test with a stricter threshold ($p < 0.01$) to avoid false positives.
- **Small samples (n < 30)**: Strictly checks normality with standard threshold ($p < 0.05$).

This approach avoids the common pitfall of automatically switching to non-parametric tests for large samples where t-test is actually more appropriate.

Variance Equality Check: When `var.equal = NULL` and t-test is selected, Levene's test is performed. If variances are unequal ($p < 0.05$), Welch's t-test is used automatically.

Visualization: The plot includes:

- Boxplot, violin plot, or both (based on `plot_type`)
- Individual data points (if `add_jitter = TRUE`)
- Statistical comparison with p-value
- Publication-ready styling

Value

An object of class `quick_ttest_result` containing:

plot A ggplot object with the comparison visualization

test_result The htest object from `t.test()` or `wilcox.test()`

method_used Character string of the test method used

normality_tests List of Shapiro-Wilk test results for each group

variance_test Levene's test result (if applicable)

descriptive_stats Data frame with descriptive statistics by group

auto_decision Details about automatic method selection

timestamp POSIXct timestamp of analysis

Important Notes

- **Two groups only**: This function requires exactly 2 levels in the grouping variable.
- **Sample size warnings**: The function will warn if sample sizes are very small (< 5) or highly unbalanced (ratio > 3:1).
- **Missing values**: Automatically removed with a warning.

See Also

`t.test`, `wilcox.test`

Examples

```
# Example 1: Basic usage with automatic method selection
set.seed(123)
data <- data.frame(
  group = rep(c("Control", "Treatment"), each = 30),
  expression = c(rnorm(30, mean = 5), rnorm(30, mean = 6))
)

result <- quick_ttest(data, group = group, value = expression)
print(result)

# Example 2: Paired samples (e.g., before/after)
paired_data <- data.frame(
  patient = rep(1:20, 2),
  timepoint = rep(c("Before", "After"), each = 20),
  score = c(rnorm(20, 50, 10), rnorm(20, 55, 10))
)

result <- quick_ttest(paired_data,
                      group = timepoint,
                      value = score,
                      paired = TRUE,
                      id = patient)

# Example 3: Non-normal data with manual method selection
skewed_data <- data.frame(
  group = rep(c("A", "B"), each = 25),
  value = c(rexp(25, rate = 0.5), rexp(25, rate = 1))
)

result <- quick_ttest(skewed_data,
                      group = group,
                      value = value,
                      method = "wilcox.test",
                      verbose = TRUE)

# Example 4: Customize visualization
result <- quick_ttest(data,
                      group = group,
                      value = expression,
                      plot_type = "both",
                      palette = "qual_balanced",
                      p_label = "p.format")

# Access components
result$plot              # ggplot object
result$test_result       # htest object
summary(result)          # Detailed summary
```

`read_excel_flex` *Flexible Excel reader*

Description

Read an Excel sheet via `readxl::read_excel()` with optional column-name cleaning (`janitor::clean_names()`), basic type control, and CLI messages.

Usage

```
read_excel_flex(
  file_path,
  sheet = 1,
  skip = 0,
  header = TRUE,
  range = NULL,
  col_types = NULL,
  clean_names = TRUE,
  guess_max = 1000,
  trim_ws = TRUE,
  na = "",
  verbose = TRUE
)
```

Arguments

`file_path`	Path to the Excel file (.xlsx or .xls).
`sheet`	Sheet name or index to read (default: 1).
`skip`	Number of lines to skip before reading data (default: 0).
`header`	Logical. Whether the first row contains column names (default: TRUE).
`range`	Optional cell range (e.g., `"B2:D100"`). Default: `NULL`.
`col_types`	Optional vector specifying column types; passed to `readxl`.
`clean_names`	Logical. Clean column names with `janitor::clean_names()` (default: TRUE).
`guess_max`	Max rows to guess column types (default: 1000).
`trim_ws`	Logical. Trim surrounding whitespace in text fields (default: TRUE).
`na`	Values to interpret as NA (default: `""`).
`verbose`	Logical. Show CLI output (default: TRUE).

Value

A tibble (or data.frame) read from the Excel sheet.

read_table_flex *Flexible and fast table reader using data.table::fread*

Description

Robust table reader with auto delimiter detection for `.csv`, `.tsv`, `.txt`, and their `.gz` variants. Uses `data.table::fread()` and prints CLI messages.

Usage

```
read_table_flex(
  file_path,
  sep = NULL,
  encoding = "UTF-8",
  header = TRUE,
  df = TRUE,
  verbose = FALSE
)
```

Arguments

`file_path`	Character. Path to the file to be read.
`sep`	Optional. Field delimiter. If `NULL`, auto-detected by file extension.
`encoding`	Character. File encoding accepted by fread: "unknown", "UTF-8", or "Latin-1".
`header`	Logical. Whether the file contains a header row. Default: `TRUE`.
`df`	Logical. Return data.frame instead of data.table. Default: `TRUE`.
`verbose`	Logical. Show progress and details. Default: `FALSE`.

Value

A `data.frame` (default) or `data.table` depending on `df` parameter.

reload_palette_cache *Reload Palette Cache*

Description

Force reload of palette data from disk. This is useful if you've updated the palette RDS file and want to refresh the cached data without restarting R.

Usage

```
reload_palette_cache()
```

Value

Invisible NULL

Examples

```
## Not run:
# After updating palettes.rds, reload the cache
reload_palette_cache()

## End(Not run)
```

`remind` *Show usage tips for common R commands*

Description

A helper to recall commonly used R functions with short examples.

Usage

```
remind(keyword = NULL)
```

Arguments

`keyword`	A keyword like "glimpse" or "read_excel". If `NULL`, show all.

Value

Invisibly returns the matched keywords (character vector).

Examples

```
remind("glimpse")
remind()  # show all keywords
```

remove_palette *Remove a Saved Palette JSON*

Description

Remove a palette file by name, trying across types if necessary.

Usage

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

Arguments

name	Character. Palette name (without '.json' suffix).
type	Character. Optional. Preferred type ("sequential", "diverging", or "qualitative").
color_dir	Character. Root folder where palettes are stored (required). Use tempdir() for examples/tests.
log	Logical. Whether to log palette removal to a temporary log file.

Value

Invisibly TRUE if removed successfully, FALSE otherwise.

Examples

```
## Not run:
# Remove a palette (requires write permissions):
remove_palette("seq_blues")

# Remove with specific type:
remove_palette("qual_vivid", type = "qualitative")

## End(Not run)
```

rgb2hex *Convert RGB values to HEX color codes*

Description

Convert an RGB triplet (or a list of triplets) to HEX color codes.

Usage

```
rgb2hex(rgb)
```

Arguments

rgb	A numeric vector of length 3 (e.g., c(255, 128, 0)), or a list of such vectors (e.g., list(c(255,128,0), c(0,255,0))).

Value

A HEX color string if a single RGB vector is provided, or a character vector of HEX codes if a list is provided.

Examples

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

---

safe_execute *Safely Execute an Expression*

Description

Evaluate code with unified error handling (and consistent warning reporting). On error, prints a CLI message (unless quiet = TRUE) and returns NULL.

Usage

```
safe_execute(expr, fail_message = "An error occurred", quiet = FALSE)
```

Arguments

expr	Code to evaluate.
fail_message	Message to display if an error occurs. Default: "An error occurred".
quiet	Logical. If TRUE, suppress messages. Default: FALSE.

Value

The result of the expression if successful; otherwise NULL.

Examples

```
safe_execute(log(1))
safe_execute(log("a"), fail_message = "Failed to compute log")
```

`scale_evanverse` *Discrete Color and Fill Scales for evanverse Palettes*

Description

Apply evanverse color palettes to ggplot2 discrete scales. These functions provide a seamless integration between evanverse palettes and ggplot2's color/fill aesthetics.

Usage

```
scale_color_evanverse(
  palette,
  type = NULL,
  n = NULL,
  reverse = FALSE,
  na.value = "grey50",
  guide = "legend",
  ...
)

scale_fill_evanverse(
  palette,
  type = NULL,
  n = NULL,
  reverse = FALSE,
  na.value = "grey50",
  guide = "legend",
  ...
)

scale_colour_evanverse(
  palette,
  type = NULL,
  n = NULL,
  reverse = FALSE,
  na.value = "grey50",
  guide = "legend",
  ...
)
```

Arguments

`palette`	Character. Name of the palette (e.g., "qual_vivid", "seq_blues"). Type will be automatically inferred from the prefix if not specified.
`type`	Character. Palette type: "sequential", "diverging", or "qualitative". If `NULL` (default), the type is automatically inferred from the palette name prefix.

n	Integer. Number of colors to use. If NULL (default), all colors from the palette are used. If n exceeds the number of colors in the palette, an error will be raised.
reverse	Logical. Should the color order be reversed? Default is FALSE.
na.value	Character. Color to use for NA values. Default is "grey50".
guide	Character or function. Type of legend. Use "legend" for standard legend or "none" to hide the legend. See guide_legend for more options.
...	Additional arguments passed to scale_color_manual or scale_fill_manual, such as name, labels, limits, etc.

Details

The scale_color_evanverse() and scale_fill_evanverse() functions automatically:

- Infer palette type from the naming convention (seq_, div_, qual_)
- Handle color interpolation intelligently based on palette type:
  - **Qualitative palettes**: Direct color selection (no interpolation)
  - **Sequential/Diverging palettes**: Smooth interpolation when n < palette size
- Support all standard ggplot2 scale parameters
- Provide informative error messages and warnings

Value

A ggplot2 scale object that can be added to a ggplot.

See Also

get_palette for retrieving palette colors, list_palettes for available palettes, scale_color_manual for the underlying ggplot2 function.

Examples

```
library(ggplot2)

# Basic usage with automatic type inference
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3, alpha = 0.8) +
  scale_color_evanverse("qual_vivid") +
  theme_minimal()

# Fill scale for boxplots
ggplot(iris, aes(x = Species, y = Sepal.Length, fill = Species)) +
  geom_boxplot(alpha = 0.7) +
  scale_fill_evanverse("qual_vivid") +
  theme_minimal()

# Reverse color order
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  scale_color_evanverse("qual_vivid", reverse = TRUE) +
```

```
  theme_minimal()

# Explicitly specify type
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  scale_color_evanverse("qual_vivid", type = "qualitative") +
  theme_minimal()

# Limit number of colors
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  scale_color_evanverse("qual_vivid", n = 3) +
  theme_minimal()

# Custom legend name and labels
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  scale_color_evanverse(
    "qual_vivid",
    name = "Iris Species",
    labels = c("Setosa", "Versicolor", "Virginica")
  ) +
  theme_minimal()

# Bar plot with fill
ggplot(mtcars, aes(x = factor(cyl), fill = factor(cyl))) +
  geom_bar() +
  scale_fill_evanverse("qual_vibrant") +
  labs(x = "Cylinders", y = "Count", fill = "Cylinders") +
  theme_minimal()
```

`set_mirror` *Set CRAN/Bioconductor Mirrors*

Description

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

Usage

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

Arguments

`repo`	Character. Repository type: "cran", "bioc", or "all" (default: "all").
`mirror`	Character. Predefined mirror name (default: "tuna").

Details

Available CRAN mirrors:

- **official**: R Project cloud server
- **rstudio**: RStudio CRAN mirror
- **tuna**: Tsinghua University (China)
- **ustc**: USTC (China)
- **aliyun**: Alibaba Cloud (China)
- **sjtu**: Shanghai Jiao Tong University (China)
- **pku**: Peking University (China)
- **hku**: Hong Kong University
- **westlake**: Westlake University (China)
- **nju**: Nanjing University (China)
- **sustech**: SUSTech (China)

Available Bioconductor mirrors:

- **official**: Bioconductor official server
- **tuna**: Tsinghua University (China)
- **ustc**: USTC (China)
- **westlake**: Westlake University (China)
- **nju**: Nanjing University (China)

Value

Previous mirror settings (invisibly).

Examples

```
## Not run:
# Set all mirrors to tuna (default):
set_mirror()

# Set only CRAN mirror:
set_mirror("cran", "westlake")

# Set only Bioconductor mirror:
set_mirror("bioc", "ustc")

# Check current settings:
getOption("repos")
getOption("BioC_mirror")

## End(Not run)
```

stat_power *Calculate Statistical Power*

Description

Compute the statistical power (probability of correctly rejecting the null hypothesis) for a given sample size, effect size, and significance level. Supports multiple test types including t-tests, ANOVA, proportion tests, correlation tests, and chi-square tests.

Usage

```
stat_power(
  n,
  effect_size,
  test = c("t.test", "anova", "proportion", "correlation", "chisq"),
  type = c("two.sample", "one.sample", "paired"),
  alternative = c("two.sided", "less", "greater"),
  alpha = 0.05,
  k = NULL,
  df = NULL,
  plot = TRUE,
  plot_range = NULL,
  palette = "qual_vivid",
  verbose = TRUE
)
```

Arguments

n	Integer. Sample size. Interpretation depends on test type: • t-tests and ANOVA: Sample size per group • Proportion test: Total sample size (one-sample test) • Correlation: Total number of paired observations • Chi-square: Total sample size
effect_size	Numeric. Effect size appropriate for the test (must be positive): • 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) • Correlation coefficient r for correlation tests (small: 0.1, medium: 0.3, large: 0.5). Use absolute value; power is the same for positive and negative correlations. • Cohen's w for chi-square tests (small: 0.1, medium: 0.3, large: 0.5)
test	Character. Type of statistical test: "t.test" (default), "anova", "proportion", "correlation", or "chisq".
type	Character. For t-tests only: "two.sample" (default), "one.sample", or "paired".

`alternative`	Character. Direction of alternative hypothesis: "two.sided" (default), "less", or "greater". Note: Only applicable to t-tests, proportion tests, and correlation tests. Ignored for ANOVA and chi-square tests (which are inherently non-directional).
`alpha`	Numeric. Significance level (Type I error rate). Default: 0.05.
`k`	Integer. Number of groups (required for ANOVA).
`df`	Integer. Degrees of freedom (required for chi-square tests).
`plot`	Logical. Generate a power curve plot? Default: TRUE.
`plot_range`	Numeric vector of length 2. Range of sample sizes for the power curve. If NULL (default), automatically determined.
`palette`	Character. evanverse palette name for the plot. Default: "qual_vivid".
`verbose`	Logical. Print detailed diagnostic information? Default: TRUE.

Details

Statistical power is the probability of correctly rejecting the null hypothesis when it is false (i.e., detecting a true effect). Conventionally, a power of 0.8 (80%) is considered adequate, though higher power (0.9 or 0.95) may be desirable in some contexts.

The function uses the **pwr** package for all power calculations, ensuring accurate results based on well-established statistical theory.

Value

An object of class `stat_power_result` containing:

power The calculated statistical power (probability of detecting the effect)

n Sample size used in the calculation

effect_size Effect size used in the calculation

alpha Significance level

test_type Type of statistical test

test_subtype Subtype for t-tests (e.g., "two.sample", "one.sample", "paired"); NULL for other tests

alternative Direction of alternative hypothesis used in the test

k Number of groups (for ANOVA); NULL for other tests

df Degrees of freedom (for chi-square tests); NULL for other tests

plot ggplot2 object showing the power curve (if plot = TRUE); NULL otherwise

pwr_object Raw result object from the **pwr** package function

details List with interpretation and recommendation text

timestamp POSIXct timestamp of when the calculation was performed

Test-Specific Notes

Proportion Test: Uses `pwr.p.test`, which is for one-sample proportion tests (testing a single proportion against a hypothesized value). For two-sample proportion tests, consider using specialized tools or packages. The `effect_size` parameter uses Cohen's h, which quantifies the difference between two proportions. In one-sample settings, Cohen's h is computed from the observed proportion p and the null hypothesis proportion $p0$.

Power Curve

When `plot = TRUE`, a power curve is generated showing how statistical power changes with sample size. The curve helps visualize:

- The current power level (marked with a red point)
- The conventional 0.8 power threshold (red dashed line)
- How increasing sample size affects power

Examples

```
## Not run:
# Example 1: Power for a two-sample t-test
result <- stat_power(
  n = 30,
  effect_size = 0.5,
  test = "t.test",
  type = "two.sample"
)
print(result)
plot(result)

# Example 2: Power for ANOVA with 3 groups
stat_power(
  n = 25,
  effect_size = 0.25,
  test = "anova",
  k = 3
)

# Example 3: Power for correlation test
stat_power(
  n = 50,
  effect_size = 0.3,
  test = "correlation"
)

## End(Not run)
```

`stat_samplesize` *Calculate Required Sample Size*

Description

Compute the minimum sample size required to achieve a specified statistical power for detecting a given effect size. Supports multiple test types including t-tests, ANOVA, proportion tests, correlation tests, and chi-square tests.

Usage

```
stat_samplesize(
  power = 0.8,
  effect_size,
  test = c("t.test", "anova", "proportion", "correlation", "chisq"),
  type = c("two.sample", "one.sample", "paired"),
  alternative = c("two.sided", "less", "greater"),
  alpha = 0.05,
  k = NULL,
  df = NULL,
  plot = TRUE,
  plot_range = NULL,
  palette = "qual_vivid",
  verbose = TRUE
)
```

Arguments

power	Numeric. Target statistical power (probability of correctly rejecting the null hypothesis). Default: 0.8 (80%).
effect_size	Numeric. Effect size appropriate for the test: • 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) • Correlation coefficient r for correlation tests (small: 0.1, medium: 0.3, large: 0.5) • Cohen's w for chi-square tests (small: 0.1, medium: 0.3, large: 0.5)
test	Character. Type of statistical test: "t.test" (default), "anova", "proportion", "correlation", or "chisq".
type	Character. For t-tests only: "two.sample" (default), "one.sample", or "paired".
alternative	Character. Direction of alternative hypothesis: "two.sided" (default), "less", or "greater".
alpha	Numeric. Significance level (Type I error rate). Default: 0.05.
k	Integer. Number of groups (required for ANOVA).
df	Integer. Degrees of freedom (required for chi-square tests).
plot	Logical. Generate a sample size curve plot? Default: TRUE.
plot_range	Numeric vector of length 2. Range of effect sizes for the curve. If NULL (default), automatically determined.
palette	Character. evanverse palette name for the plot. Default: "qual_vivid".
verbose	Logical. Print detailed diagnostic information? Default: TRUE.

Details

Sample size estimation is a critical step in research planning. This function calculates the minimum number of participants needed to achieve a specified statistical power (typically 0.8 or 80%) for detecting an effect of a given size.

The function uses the **pwr** package for all calculations, ensuring accurate results based on well-established statistical theory. Sample sizes are always rounded up to the nearest integer.

Value

An object of class `stat_samplesize_result` containing:

n Required sample size (per group for t-tests and ANOVA)

n_total Total sample size across all groups

power Target statistical power

effect_size Effect size used in the calculation

alpha Significance level

test_type Type of statistical test

plot ggplot2 object showing the sample size curve (if plot = TRUE)

details List with interpretation and recommendations

Sample Size Curve

When `plot = TRUE`, a sample size curve is generated showing how required sample size changes with effect size. The curve helps visualize:

- The current required sample size (marked with a red point)
- Reference lines for small, medium, and large effects
- How detecting smaller effects requires larger samples

Important Notes

- Sample sizes are calculated per group for t-tests and ANOVA
- Consider adding 10-15% to account for potential dropout
- Very small effect sizes may require impractically large samples

See Also

`stat_power` for calculating statistical power.

Examples

```
## Not run:
# Example 1: Sample size for a two-sample t-test
result <- stat_samplesize(
  power = 0.8,
  effect_size = 0.5,
```

```
  test = "t.test",
  type = "two.sample"
)
print(result)
plot(result)

# Example 2: Sample size for ANOVA with 3 groups
stat_samplesize(
  power = 0.8,
  effect_size = 0.25,
  test = "anova",
  k = 3
)

# Example 3: Sample size for correlation test
stat_samplesize(
  power = 0.9,
  effect_size = 0.3,
  test = "correlation"
)

## End(Not run)
```

`summary.quick_chisq_result` *Summary Method for quick_chisq_result*

Description

Summary Method for quick_chisq_result

Usage

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

Arguments

`object`	An object of class `quick_chisq_result`
`...`	Additional arguments (unused)

summary.quick_cor_result *Summary method for quick_cor_result*

Description

Summary method for quick_cor_result

Usage

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

Arguments

object	A quick_cor_result object
...	Additional arguments (unused)

summary.quick_ttest_result *Summary method for quick_ttest_result*

Description

Summary method for quick_ttest_result

Usage

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

Arguments

object	A quick_ttest_result object
...	Additional arguments (unused)

trial *Clinical Trial Dataset*

Description

A sample clinical trial dataset used for testing and demonstration of data analysis functions. Contains typical clinical trial variables for testing various statistical and visualization functions.

Format

A data frame with 200 rows and 8 columns:

trt Character vector of treatment assignments

age Numeric vector of patient ages

marker Numeric vector of biomarker levels

stage Factor with tumor stage levels

grade Factor with tumor grade levels

response Integer vector indicating tumor response

death Integer vector indicating patient death

ttdeath Numeric vector of time to death/censoring

Source

Created for testing and demonstration purposes.

update_pkg *Update R Packages*

Description

Update R packages from CRAN, GitHub, or Bioconductor. Supports full updates, source-specific updates, or targeted package updates. Automatically handles version compatibility checks and respects mirror settings.

Usage

```
update_pkg(pkg = NULL, source = NULL, ...)
```

Arguments

pkg	Character vector. Package name(s) to update. For GitHub, use "user/repo" format. Only required when source is specified.
source	Character. Package source: "CRAN", "GitHub", or "Bioconductor". Optional if updating all installed CRAN and Bioconductor packages.
...	Additional arguments passed to install.packages, install_github, or install.

Value

NULL (invisibly). Side effect: updates packages.

Examples

```
## Not run:
# Update all CRAN + Bioconductor packages:
update_pkg()

# Update all CRAN packages only:
update_pkg(source = "CRAN")

# Update specific package:
update_pkg("ggplot2", source = "CRAN")

# Update GitHub package:
update_pkg("hadley/ggplot2", source = "GitHub")

## End(Not run)
```

---

view *Quick interactive table viewer (reactable)*

Description

Quickly view a data.frame or tibble as an interactive table in the Viewer pane.

Usage

```
view(
  data,
  page_size = 10,
  searchable = TRUE,
  filterable = TRUE,
  striped = TRUE,
  highlight = TRUE,
  compact = FALSE
)
```

Arguments

data	A data.frame or tibble to display.
page_size	Number of rows per page (default = 10).
searchable	Whether to enable search (default = TRUE).
filterable	Whether to enable column filters (default = TRUE).
striped	Whether to show striped rows (default = TRUE).

highlight	Whether to highlight rows on hover (default = TRUE).
compact	Whether to use a compact layout (default = FALSE).

Value

A reactable widget rendered in the Viewer pane.

Examples

```
if (requireNamespace("reactable", quietly = TRUE)) {
  view(iris)
  view(mtcars, page_size = 20, striped = TRUE, filterable = TRUE)
}
```

---

void *Void Value Utilities*

Description

A comprehensive suite of functions for detecting, removing, and managing "void" values (NA, NULL, and empty strings) in R objects.

Usage

```
is_void(x, include_na = TRUE, include_null = TRUE, include_empty_str = TRUE)

any_void(x, include_na = TRUE, include_null = TRUE, include_empty_str = TRUE)

drop_void(x, include_na = TRUE, include_null = TRUE, include_empty_str = TRUE)

replace_void(
  x,
  value = NA,
  include_na = TRUE,
  include_null = TRUE,
  include_empty_str = TRUE
)

cols_with_void(
  data,
  include_na = TRUE,
  include_null = TRUE,
  include_empty_str = TRUE,
  return_names = TRUE
)

rows_with_void(
```

```
  data,
  include_na = TRUE,
  include_null = TRUE,
  include_empty_str = TRUE
)
```

Arguments

x	A vector or list.
include_na	Logical. Detect NA if TRUE. Default: TRUE.
include_null	Logical. Detect NULL if TRUE. Default: TRUE.
include_empty_str	Logical. Detect empty strings "" if TRUE. Default: TRUE.
value	The replacement value to use for voids. Default: NA.
data	A data.frame or tibble.
return_names	Logical. If TRUE (default), return column names; else logical vector.

Details

The void utilities family consists of:

- is_void: Core detection function returning logical vector
- any_void: Check if any void value exists
- drop_void: Remove void values from vectors/lists
- replace_void: Replace void values with custom values
- cols_with_void: Detect columns containing void values
- rows_with_void: Detect rows containing void values

All functions support customizable void detection through three parameters:

- include_na: Consider NA as void (default: TRUE)
- include_null: Consider NULL as void (default: TRUE)
- include_empty_str: Consider "" as void (default: TRUE)

Value

A logical vector indicating which elements are void.

- If x is NULL, returns a single TRUE (if include_null=TRUE) or FALSE.
- If x is an empty vector, returns logical(0).
- If x is a list, evaluates each element recursively and returns a flattened logical vector.
- For atomic vectors, returns a logical vector of the same length.

A single logical value:

- TRUE if any void values are present.

- FALSE otherwise.
- For NULL input, returns TRUE if include_null = TRUE, else FALSE.

A cleaned vector or list of the same type as input, with void values removed.

A cleaned vector or list with void values replaced.

A character vector (column names) or logical vector indicating void presence per column.

A logical vector of length nrow(data) indicating whether each row contains at least one void value.

is_void()

Check for Null / NA / Blank ("") Values

Determine whether input values are considered "void": NULL, NA, or "". Each condition is controlled by a dedicated argument.

any_void()

Check if Any Value is Void (NA / NULL / "")

Test whether any element in a vector or list is considered "void". Void values include NA, NULL, and empty strings (""), and you can customize which ones to consider.

drop_void

Remove Void Values from a Vector or List

Removes elements from a vector or list that are considered "void": NA, NULL, and empty strings (""). Each can be toggled via parameters.

replace_void

Replace void values (NA / NULL / "")

Replace elements in a vector or list considered "void" with a specified value. Void values include NA, NULL, and empty strings "" (toggle via flags).

cols_with_void()

Detect Columns Containing Void Values

Scan a data.frame or tibble and identify columns that contain any "void" values. Void values include NA, NULL, and "", which can be toggled via parameters.

rows_with_void

Detect rows containing void values (NA / NULL / "")

Scan a data.frame or tibble and identify rows that contain any "void" values. Void values include NA, NULL, and empty strings "" (toggle via flags).

Examples

```
is_void(c(NA, "", "text"))                  # TRUE TRUE FALSE
is_void(list(NA, "", NULL, "a"))            # TRUE TRUE TRUE FALSE
is_void("NA", include_na = FALSE)           # FALSE
is_void(NULL)                               # TRUE
any_void(c("a", "", NA))                # TRUE
any_void(list("x", NULL, "y"))          # TRUE
any_void(c("a", "b", "c"))              # FALSE
any_void(NULL)                          # TRUE
any_void("", include_empty_str = FALSE) # FALSE
drop_void(c("apple", "", NA, "banana"))
drop_void(list("A", NA, "", NULL, "B"))
drop_void(c("", NA), include_na = FALSE)
replace_void(c(NA, "", "a"), value = "N/A")
replace_void(list("A", "", NULL, NA), value = "missing")
replace_void(c("", "b"), value = 0, include_empty_str = TRUE)
df <- data.frame(name = c("A", "", "C"), score = c(1, NA, 3), id = 1:3)
cols_with_void(df)
cols_with_void(df, return_names = FALSE)
cols_with_void(df, include_na = FALSE)
df <- data.frame(id = 1:3, name = c("A", "", "C"), score = c(10, NA, 20))
rows_with_void(df)
df[rows_with_void(df), ]
```

`with_timer` *Wrap a function to measure and display execution time*

Description

Wraps a function with CLI-based timing and prints its runtime in seconds. Useful for benchmarking or logging time-consuming tasks.

Usage

```
with_timer(fn, name = "Task")
```

Arguments

`fn`	A function to be wrapped.
`name`	A short descriptive name of the task (used in log output).

Details

Requires the `tictoc` package (CLI messages are emitted via `cli`).

Value

A function that executes `fn(...)` and prints timing information (returns invisibly).

Examples

```
slow_fn <- function(n) { Sys.sleep(0.01); n^2 }
timed_fn <- with_timer(slow_fn, name = "Square Task")
timed_fn(5)
```

---

write_xlsx_flex *Flexible Excel writer*

Description

Write a data frame or a **named** list of data frames to an Excel file with optional styling.

Usage

```
write_xlsx_flex(
  data,
  file_path,
  overwrite = TRUE,
  timestamp = FALSE,
  with_style = TRUE,
  auto_col_width = TRUE,
  open_after = FALSE,
  verbose = TRUE
)
```

Arguments

data	A data.frame, or a **named** list of data.frames.
file_path	Output path to a .xlsx file.
overwrite	Whether to overwrite if the file exists. Default: TRUE.
timestamp	Whether to append a date suffix (YYYY-MM-DD) to the filename. Default: FALSE.
with_style	Whether to apply a simple header style (bold, fill, centered). Default: TRUE.
auto_col_width	Whether to auto-adjust column widths. Default: TRUE.
open_after	Whether to open the file after writing (platform-dependent). Default: FALSE.
verbose	Whether to print CLI messages (info/warn/success). Errors are always shown. Default: TRUE.

Value

No return value; writes a file to disk.

%is% *Strict identity comparison with diagnostics*

Description

A semantic operator that checks whether two objects are strictly identical, and prints where they differ if not.

Usage

```
a %is% b
```

Arguments

a	First object (vector, matrix, or data.frame)
b	Second object (vector, matrix, or data.frame)

Value

TRUE if identical, FALSE otherwise (with diagnostics)

Examples

```
1:3 %is% 1:3                 # TRUE
1:3 %is% c(1, 2, 3)          # FALSE, type mismatch (integer vs double)
data.frame(x=1) %is% data.frame(y=1)  # FALSE, column name mismatch
m1 <- matrix(1:4, nrow=2)
m2 <- matrix(c(1,99,3,4), nrow=2)
m1 %is% m2                  # FALSE, value differs at [1,2]
c(a=1, b=2) %is% c(b=2, a=1) # FALSE, names differ
```

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

Description

Performs case-insensitive matching between elements in `x` and entries in `table`, returning a named character vector: names are the matched entries from `table`, values are the original elements from `x`. Unmatched values are ignored (not included in the result).

Usage

```
x %map% table
```

Arguments

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

Value

A named character vector. Names are from matched table values, values are from x. If no matches are found, returns a zero-length named character vector.

Examples

```
# Basic matching (case-insensitive)
c("tp53", "brca1", "egfr") %map% c("TP53", "EGFR", "MYC")
# returns: Named vector: TP53 = "tp53", EGFR = "egfr"

# Values not in table are dropped
c("akt1", "tp53") %map% c("TP53", "EGFR")
# returns: TP53 = "tp53"

# All unmatched values returns: empty result
c("none1", "none2") %map% c("TP53", "EGFR")
# returns: character(0)
```

%match% *%match%: Case-insensitive match returning indices*

Description

Performs case-insensitive matching, like base::match(), but ignores letter case.

Usage

```
x %match% table
```

Arguments

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

Value

An integer vector of the positions of matches of x in table, like base::match(). Returns NA for non-matches. Returns an integer(0) if x is length 0.

Examples

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

# No matches returns: all NA
c("aaa", "bbb") %match% c("xxx", "yyy")

# Empty input
character(0) %match% c("a", "b")

# Order sensitivity (like match): first match is returned
c("x") %match% c("X", "x", "x")
# returns: 1
```

`%nin%` `%nin%`*: Not-in operator (negation of* `%in%`*)*

Description

A binary operator to test whether elements of the left-hand vector are **not** present in the right-hand vector. This is equivalent to `!(x %in% table)`.

Usage

```
x %nin% table
```

Arguments

`x`	vector or NULL: the values to be matched.
`table`	vector or NULL: the values to be matched against.

Value

A logical vector where `TRUE` indicates the corresponding element of `x` is not present in `table`. Results involving `NA` follow base R semantics: e.g., if `x` contains `NA` and `table` does not, the result at that position is `NA` (since `!NA` is `NA`).

Examples

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

# Works with mixed types as `%in%` does:
c(1, "a") %nin% c("a", "b", 2)
```

%p% *%p%: paste two strings with a single space*

Description

An infix operator for string concatenation with one space between `lhs` and `rhs`. Inspired by the readability of `%>%`, intended for expressive text building.

Usage

```
lhs %p% rhs
```

Arguments

`lhs`	A character vector on the left-hand side.
`rhs`	A character vector on the right-hand side.

Value

A character vector, concatenating `lhs` and `rhs` with a single space.

Examples

```
"Hello" %p% "world"
"Good" %p% "job"
c("hello", "good") %p% c("world", "morning")   # vectorized
```

Index