Package: leo.basic (via r-universe)

May 19, 2026

Title leo.basic

Version 0.0.2

Description Basic function for leo universe.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports cli, coin, dplyr, ggbeeswarm, ggplot2, ggprism, ggsci, glue, grDevices, magrittr, nortest, rlang, scales, stats, tibble

Suggests clusterProfiler, data.table, ggpie, ggrastr, org.Hs.eg.db, pak, patchwork, ReactomePA, testthat (>= 3.0.0), vroom

Config/testthat/edition 3

Remotes github::showteeth/ggpie

Repository https://laleoarrow.r-universe.dev

Date/Publication 2026-05-19 16:05:14 UTC

RemoteUrl https://github.com/laleoarrow/leo.basic

RemoteRef HEAD

RemoteSha 9625ce5012db7f15c6b43b623243566cf5cce75a

Contents

`enrichment_helper` *Enrichment helpers*

Description

Utilities to format gene lists and collapse IDs for KEGG/Reactome GSEA/ORA.

- You can download gmt file from https://maayanlab.cloud/Enrichr/#libraries
- Tutorial: https://yulab-smu.top/biomedical-knowledge-mining-book/faq.html#genelist

Build a named numeric vector sorted decreasingly

Format enrichment gene string like "A/B/C" to vector

Map SYMBOL-named vector to ENTREZID and keep 1 score/gene by slice_max(|score|)

Usage

```
format_geneList(df)

format_gene_str2vec(x, sep = "/", unique_only = TRUE)

prep_GSEA_geneList(geneList_sym)
```

Arguments

`df`	A data frame with two column: *Column 1*: 1st column is **Gene ID** *Column 2*: 2nd column is **Gene Statistics** (e.g., fold change or other numerical variable)
`x`	A character vector or a list of character vectors
`sep`	Separator, default "/"
`unique_only`	Return unique identifiers, default TRUE
`geneList_sym`	Named numeric vector; names are SYMBOL; values are scores

Details

Enrichment helpers

Value

A named numeric vector (descending)

A character vector or a list of character vectors

Named numeric vector whose names are unique ENTREZID (descending)

Functions

- `format_geneList`: make a named numeric vector sorted decreasingly
- `format_gene_str2vec`: split core_enrichment string into gene vector
- `prep_GSEA_geneList`: map names to ENTREZID and keep 1 score/gene by |score| max

See Also

clusterProfiler, ReactomePA

Examples

```
# ego4@result$core_enrichment[1] -> "IGLV3-19/IGHM/IGLV7-43/HLA-DPA1/IGHV3-7"
# format_gene_str2vec(ego4@result$core_enrichment[1])
## Not run:
data(geneList, package = "DOSE")
gl_sym <- stats::setNames(as.numeric(geneList), names(geneList)) # demo only
gl_entrez <- prep_GSEA_geneList(gl_sym)

## End(Not run)
```

`enrichment_individual` *Enrichment wrappers (individual)*

Description

ORA/GSEA wrappers for GO, KEGG, KEGG Module, and Reactome.

Usage

```
ORA_GO(gene, simplify = TRUE)

GSEA_GO(geneList, simplify = TRUE)

ORA_KEGG(gene, input = "SYMBOL")

GSEA_KEGG(geneList, input = "SYMBOL")
```

```
ORA_MKEGG(gene, input = "SYMBOL")

GSEA_MKEGG(geneList, input = "SYMBOL")

ORA_Reactome(gene, input = "SYMBOL")

GSEA_Reactome(geneList, input = "SYMBOL")
```

Arguments

gene	SYMBOL or ENTREZID vector depending on input
simplify	Use clusterProfiler::simplify for GO terms
geneList	Named numeric vector (SYMBOL or ENTREZID per input)
input	"SYMBOL" or "ENTREZID"

Details

Enrichment wrappers (individual)

Value

enrichResult

gseaResult

enrichResult

gseaResult

enrichResult

gseaResult

enrichResult

gseaResult

Functions

- ORA_GO, GSEA_GO
- ORA_KEGG, GSEA_KEGG
- ORA_MKEGG, GSEA_MKEGG
- ORA_Reactome, GSEA_Reactome

See Also

leo_enrich

Examples

```
## Not run:
# KEGG vis notes:
# https://yulab-smu.top/biomedical-knowledge-mining-book/clusterprofiler-kegg.html
library("pathview")
kk <- gseKEGG(geneList     = geneList,
              keyType      = 'SYMBOL',
              organism     = 'hsa',
              minGSSize    = 120,
              pvalueCutoff = 0.05,
              verbose      = FALSE)
browseKEGG(kk, 'hsa04110')
# or
hsa04110 <- pathview(
  gene.data  = geneList,
  pathway.id = "hsa04110",
  species    = "hsa",
  limit      = list(gene = max(abs(geneList)), cpd = 1)
)

## End(Not run)
```

`install_deps` *Install LEO package dependencies*

Description

Install dependency sets for `leo.basic` or other `leo.*` packages universe using `pak`.

Usage

```
install_deps(leo.pak = "leo.basic", ncpus = 4L, upgrade = FALSE)
```

Arguments

`leo.pak`	Character scalar. LEO package name. Default `"leo.basic"`.
`ncpus`	Integer. Number of CPU cores for parallel compilation. Default 4.
`upgrade`	Logical. Whether to upgrade already installed packages. Default FALSE.

Value

Invisibly returns the package specs passed to `pak::pkg_install()`.

Examples

```
## Not run:
# 1) Install pak
install.packages("pak", repos = "https://cloud.r-project.org")

# 2) Install leo.basic first
pak::pkg_install("laleoarrow/leo.basic")

# 3) Install leo.ukb (deps first, then package)
leo.basic::install_deps("leo.ukb", ncpus = 8)
pak::pkg_install("laleoarrow/leo.ukb")

# 4) Install leo.gwas (deps first, then package)
leo.basic::install_deps("leo.gwas", ncpus = 8)
pak::pkg_install("laleoarrow/leo.gwas")

## End(Not run)
```

---

install_local *Install local package*

Description

This function is writen as I often forget what to code when I want to install a package from local source. This now uses `pak::pkg_install()` for a consistent installation backend. Anyway, it is always good to simplify the process.

Usage

```
install_local(path)
```

Arguments

`path`	The path to the local package source.

---

leo_discrete_color *Generate a discrete color palette (expanded from a base panel)*

Description

Return `n` distinct colors or a named vector for `levels`. Uses a fixed base panel and expands smoothly if more colors are needed.

Usage

```
leo_discrete_color(levels = NULL, n = NULL, base_panel = NULL)
```

Arguments

levels	character; category names. If provided, output is named with levels. If NULL, use n.
n	integer; number of colors to return when levels is NULL.
base_panel	named character; optional seed palette (hex). Defaults to an internal panel.

Value

character vector of hex colors; named if levels is provided.

Examples

```
leo_discrete_color(n = 8)
leo_discrete_color(levels = c("Brain","Liver","Heart"))
```

---

leo_enrich *Enrichment analysis (Integrated)*

Description

Enrichment analysis for GO/KEGG/MKEGG/Reactome using ORA or GSEA; iterate method first, then background.

Usage

```
leo_enrich(
  gene,
  geneList,
  simplify = TRUE,
  input = "SYMBOL",
  method = c("ORA", "GSEA"),
  background = c("GO", "KEGG", "MKEGG", "Reactome")
)
```

Arguments

gene	SYMBOL vector for ORA
geneList	Named numeric vector for GSEA (names match input)
simplify	Whether to simplify GO results
input	"SYMBOL" or "ENTREZID" (for KEGG/MKEGG/Reactome)
method	c("ORA","GSEA"), supports multiple
background	c("GO","KEGG","MKEGG","Reactome"), supports multiple

Value

A list of enrichment results per method-background combo

Examples

```
## Not run:
# We here use SYMBOL in the first place
library(org.Hs.eg.db)
data(geneList, package="DOSE")
gene.df <- tibble(logFC=geneList, ID=names(geneList))
gene.df.map <- bitr(gene.df$ID, fromType = "ENTREZID", # optional
                    toType = c("SYMBOL"),
                    OrgDb = org.Hs.eg.db)
gene.df <- gene.df %>% left_join(gene.df.map, by = c("ID" = "ENTREZID")) %>% drop_na()
gene.df <- gene.df %>% arrange(desc(logFC))
gene <- gene.df$SYMBOL[1:100] # for ORA
geneList <- setNames(gene.df$logFC, gene.df$SYMBOL)  # for GSEA

## End(Not run)
```

`leo_log` *Log messages with time stamps*

Description

Logs messages with time stamps The messages are styled using the `cli` package for enhanced readability. This function can not deal with function in the `cli` package.

Usage

```
leo_log(..., level = "info", verbose = TRUE)
```

Arguments

`...`	The message string to log, which will be pasted together.
`level`	The log level. Options are `"info"`, `"success"`, `"warning"`, and `"danger"`.
`verbose`	in case you want to turn the info log off, set it to FALSE.

Value

No return value. Outputs a formatted log message with a timestamp.

Examples

```
n1 <- 10; n2 <- 20
leo_log("Processing the", n1, "and", n2, "files.")
leo_log("Task completed successfully!", level = "success")
leo_log("Potential issue detected.", level = "warning")
leo_log("Error occurred during processing!", level = "danger")
```

leo_message *Give messages with my color*

Description

Give messages with my color

Usage

```
leo_message(..., color = "31", return = FALSE)
```

Arguments

`...`	The messages you wanna messgae, which will be pasted together
`color`	Str. Preferred color. Default is yellow. Options are "31" (red), "32" (green), "34" (blue), "95" (light purple)...
`return`	Logical. If TRUE, returns the formatted string. If FALSE (Default), prints directly.

Examples

```
leo_message("This is a red message", color = "31")
leo_message("This is a green message", "\nhaha", color = "32")
leo_message("This is a blue ", "message", color = "34")
leo_message("This is a light purple message", color = "95")
leo_message("Welcome to use the LEO package!")
```

leo_more_color *Generate a custom colour palette*

Description

Convenience wrapper around `colorRampPalette()` that follows my-preferred coding conventions.

Usage

```
leo_more_color(color_set, expected_num)
```

Arguments

`color_set`	**Character vector.** Base colours to interpolate between (e.g. `c("#1f77b4", "#ff7f0e", "#2ca02c")`).
`expected_num`	**Integer (scalar).** Number of colours you want to generate.

Value

A character vector of hexadecimal colour codes of length `expected_num`.

Examples

```
# 10-colour palette smoothly blended from three anchor colours
leo_more_color(c("#1f77b4", "#ff7f0e", "#2ca02c"), 10)
```

`leo_permutation_test` *leo_permutation_test: flexible one-way permutation / rank-based testing*

Description

Compare one or multiple numeric features between two groups using permutation-based tests from **coin**. Two usage modes:

1. **Vector mode** - supply a numeric vector `data` and a grouping vector `class_vec`.
2. **Data-frame mode** - supply a data.frame `data` plus the column name holding the groups (`class_col`) and, optionally, the feature columns (`feature_cols`).

Workflow per feature:

- Normality test - Shapiro-Wilk for $n <= 5000$, Anderson-Darling for $n > 5000$.
- Variance homogeneity - F-test.
- If both criteria hold, run `coin::oneway_test`; otherwise run `coin::wilcox_test`.

Usage

```
leo_permutation_test(
  data,
  class_vec = NULL,
  class_col = "Class",
  feature_cols = NULL,
  set_seed = 725
)
```

Arguments

`data`	Numeric vector *or* data.frame.
`class_vec`	Grouping vector when `data` is numeric (length must match `data`); accepted as 0/1 or any two-level factor/character.
`class_col`	Character. Column name of grouping variable in a data.frame. Default `"Class"`.
`feature_cols`	Character vector of feature columns in the data.frame. If `NULL` (default), all numeric columns except `class_col` are used.
`set_seed`	Integer random seed (passed to `set.seed`). Default `725`.

Value

A tibble with three columns:

Feature Feature name

P P-value

Type Test used: `"oneway"` or `"wilcox"`

Examples

```
## vector mode ------------------------------------------------------------
set.seed(1)
feat <- rnorm(100); cls <- rep(c(0, 1), each = 50)
leo_permutation_test(feat, class_vec = cls)

## data-frame mode: specify feature columns -------------------------------
df <- data.frame(Class = cls,
                 F1 = feat,
                 F2 = rnorm(100, 0.3),
                 F3 = rnorm(100, 1))
leo_permutation_test(df, class_col = "Class", feature_cols = c("F1", "F2", "F3"))

## data-frame mode: all numeric features except Class ----------------------
leo_permutation_test(df, class_col = "Class")

## large-n example triggers AD test ---------------------------------------
big_feat <- rnorm(6000); big_grp <- rep(c(0, 1), each = 3000)
leo_permutation_test(big_feat, class_vec = big_grp)
```

`leo_stop` *Stop execution with formatted message*

Description

Combines `leo_log(..., level = "danger")` with `stop()` for consistent error handling. Prints a timestamped red message before stopping execution.

Usage

```
leo_stop(..., call. = FALSE)
```

Arguments

`...`	Messages to combine (passed to `leo_log`)
`call.`	Logical; if FALSE (default), the call is not included in error message

Value

This function never returns; it stops execution with an error.

Examples

```
## Not run:
if (!is.data.frame(df)) leo_stop("df must be a data.frame.")

## End(Not run)
```

`leo_theme` *Custom ggplot2 theme*

Description

I often forget my favorite ggplot theme settings, so I write this to reminds myself.

Usage

```
leo_theme(
  plot_title_size = 14,
  legend_title_size = 10,
  legend_text_size = 9,
  axis_title_size = 10,
  axis_text_size = 8,
  type = c("console", "object")
)
```

Arguments

`plot_title_size`	Numeric; title font size (default 14).
`legend_title_size`	Numeric; legend title font size (default 10).
`legend_text_size`	Numeric; legend text size (default 9).
`axis_title_size`	Numeric; axis title size (default 10).
`axis_text_size`	Numeric; axis text size (default 8).
`type`	Character; "object" to return theme, or "console" (default) to print copyable code.

Examples

```
library(ggplot2)
# Add to a plot
ggplot(mtcars, aes(wt, mpg)) + geom_point() +
  leo_theme()
# Print theme code
leo_theme(type = "console")
```

leo_time_elapsed *Intelligent Elapsed-Time Formatter*

Description

Compute and format the time difference between **now** and `start_time`, choosing the most appropriate unit automatically:

- `< 1 s` -> **milliseconds (ms)**
- `< 60 s` -> **seconds (sec)**
- `< 1 h` -> **minutes (min)**
- `< 1 day` -> **hours (hr)**
- `< 1 week` -> **days (day)**
- `< 1 month`-> **weeks (wk)**
- `< 1 year` -> **months (mon)**
- otherwise -> **years (yr)**

Usage

```
leo_time_elapsed(start_time, digits = 2, return = FALSE)
```

Arguments

`start_time`	POSIXct. output from `Sys.time()`.
`digits`	Integer. digits to keep (defaut: 2).
`return`	Logical. if TRUE, only return the calculated results.

Value

If `return = FALSE` (default), prints the elapsed time in a formatted string

Examples

```
t0 <- Sys.time() - 0.123
leo_time_elapsed(t0) # -> [12:34:56] Elapsed 125 ms

t1 <- Sys.time() - 3.5
leo_time_elapsed(t1) # -> [12:35:00] Elapsed 3.5 sec

# Capture string without printing
t2 <- Sys.time() - 61
elapsed_str <- leo_time_elapsed(t2, return = TRUE)
print(elapsed_str) # "1.08 min"
```

plot_group_numbers *Plot numeric values by group with jitter / beeswarm, color / shape / size mapping*

Description

Draw a grouped scatter of numeric values with optional jitter / beeswarm arrangement, color / shape / size encoding, plus optional mean indicator.

Usage

```
plot_group_numbers(
  df,
  group,
  number,
  color_col = NULL,
  color_rule = NULL,
  shape_col = NULL,
  shape_rule = NULL,
  size_col = NULL,
  size_rule = NULL,
  jitter = c("no", "yes", "bee"),
  x_axis_pos = c("default", "zero"),
  mean_type = c("none", "point", "line"),
  legend = TRUE
)
```

Arguments

df	data.frame.
group	column name (chr) mapped to x-axis categories.
number	column name (chr) mapped to y-axis numeric values.
color_col	column name (chr) for point color (skip color_rule).
color_rule	function(df) -> color vector; ignored if color_col given.
shape_col	column name (chr) for point shape (skip shape_rule).
shape_rule	function(df) -> shape vector; ignored if shape_col given.
size_col	column name (chr) for point size (skip size_rule).
size_rule	function(df) -> size vector; ignored if size_col given.
jitter	one of "no", "yes", "bee".
x_axis_pos	"default" or "zero" (draw baseline at y = 0).
mean_type	"none", "point", "line".
legend	Logical. Whether to show legends for colour/shape/size mappings. Default TRUE. Set FALSE to hide all legends.

Value

ggplot object.

Note

Shape tips (ggplot shape codes): Run `plot(0:25, pch = 0:25); text(0:25, labels = 0:25, pos = 3)` to preview every shape code.

Code	Marker	Visual meaning	Best use
24	solid up-triangle	solid triangle pointing up	filled point with border; use `stroke` for outline width
25	solid down-triangle	solid triangle pointing down	filled point with border; use `stroke` for outline width
22	solid square	solid square	filled point with border; use `stroke` for outline width
0	open square	open square	outline-only marker
2	open up-triangle	open triangle pointing up	outline-only marker
6	open down-triangle	open triangle pointing down	outline-only marker
3	+	plus sign	thin marker for dense overlaps
4	x	x/cross sign	thin marker for dense overlaps
1	open circle	open circle	thin marker for dense overlaps

Shapes 21-25 accept separate border (`colour`) and interior (`fill`). For filled point with black outline: shape = 21, `colour = "black"`, `fill = "<fill>"`, tweak `stroke`. Provide *_col or *_rule, or leave both NULL. Defaults: colour sig/pos=red, sig/neg=blue, else grey; shape 16; size 2.

Examples

```
set.seed(1)
df <- data.frame(
  var        = paste0("gene", 1:60),
  group      = sample(c("Tcell","Bcell","NK"), 60, TRUE),
  aver_logFC = rnorm(60, 0, 2),
  p_val      = runif(60),
  my_shape   = sample(c(24,25,22), 60, TRUE),
  my_size    = sample(c(2,3,4), 60, TRUE)
)
plot_group_numbers(df,
                   group      = "group",
                   number     = "aver_logFC",
                   shape_col  = "my_shape",
                   size_col   = "my_size",
                   jitter     = "bee",
                   x_axis_pos = "zero",
                   mean_type  = "point")
```

plot_pie *Plot pie/ring via ggpie with install hint*

Description

Plot pie/ring via ggpie with install hint

Usage

```
plot_pie(
  x,
  colors = NULL,
  color_alpha = 1,
  type = c("num", "ratio"),
  ring_ratio = 1,
  border = TRUE,
  annotation_type = c("in", "out"),
  label_type = c("horizon", "circle", "none"),
  label_size = 4
)
```

Arguments

x	Numeric vector: values (type='num') or ratios (type='ratio').
colors	Vector of fill colors; NULL uses package palette.
color_alpha	Alpha for colors in 0-1.
type	"num" or "ratio" for the meaning of x.
ring_ratio	Visible ring thickness in 0-1; 1=full pie (no hole).
border	Logical; draw slice borders or not (default TRUE).
annotation_type	"in" or "out" for label position.
label_type	"horizon" or "circle" for label style. Or "none" for no labels.
label_size	Label size.

Value

ggplot object.

Examples

```
plot_pie(
  c(A = 30, B = 20, C = 50),
  type = "num", ring_ratio = 1,
  annotation_type = "in"
)
```

```
plot_pie(
  c(A = 0.2, B = 0.3, C = 0.5),
  type = "ratio", ring_ratio = 0.6,
  annotation_type = "out",
  label_type = "horizon",
  colors = c("#66c2a5", "#fc8d62", "#8da0cb")
)
plot_pie(
  c(A = 30, B = 20, C = 50),
  ring_ratio = .6, border = FALSE,
  color_alpha = 0.8, type = "num",
  label_type = "horizon",
  annotation_type = "out"
)
```

`plot_prism_lollipop` *Prism-style lollipop plot (grouped or single series)*

Description

Create a lollipop plot with Prism-style x-axis brackets, suitable for grouped or single-series data.

Usage

```
plot_prism_lollipop(
  df,
  x_var,
  y_var,
  group_var = NULL,
  color_palette = NULL,
  y_label = "Value",
  y_as_percent = FALSE,
  plot_title = NULL,
  segment_width = 1,
  point_size = 4,
  point_stroke = 1
)
```

Arguments

`df`	Data frame containing the data.
`x_var`	Bare column name for the x-axis (categorical variable).
`y_var`	Bare column name for the y-axis (numeric variable).
`group_var`	Optional bare column name for grouping; if NULL, a single series is plotted.
`color_palette`	Named vector of colors for each group; for single-series, first color is used.
`y_label`	Label for the y-axis.

y_as_percent	Logical; if TRUE, scales y_var by 100 and formats labels as percentages.
plot_title	Optional plot title.
segment_width	Numeric; line width of the lollipop segments. Default is 1.
point_size	Numeric; size of the lollipop points. Default is 4.
point_stroke	Numeric; stroke width of point borders. Default is 1.

Value

A ggplot2 object.

Examples

```
# Single-series example
df_single <- data.frame(Stage = LETTERS[1:5], Rate = c(0.12, 0.14, 0.2, 0.17, 0.22))
plot_prism_lollipop(
  df_single, Stage, Rate,
  color_palette = c("#0072B2"),
  y_label = "Conversion Rate (%)",
  y_as_percent = TRUE,
  plot_title = "Single Series Lollipop Plot"
)

# Grouped example
df_grouped <- data.frame(Category = rep(LETTERS[1:5], 2), Score = runif(10, 0.3, 0.6),
                         Group = rep(c("G1", "G2"), each = 5))
pal <- c("G1" = "#E69F00", "G2" = "#56B4E9")
plot_prism_lollipop(
  df_grouped, Category, Score, Group,
  color_palette = pal,
  y_label = "Score (%)",
  y_as_percent = TRUE,
  plot_title = "Grouped Lollipop Plot"
)
```

---

put_legend_inside *Put legend inside ggplot/patchwork at (0.8, 0.8) npc*

Description

Works for a single ggplot, a list of ggplots, or a patchwork object.

Usage

```
put_legend_inside(p, x = 0.8, y = 0.8)
```

Arguments

p	ggplot object, list of ggplot objects, or patchwork
x, y	npc coordinates in 0-1, default 0.8, 0.8

Value

Same type as input with legend repositioned

`rasterize_layers` *Rasterize point-like layers in a ggplot object*

Description

Rasterize point-like layers in a ggplot object

Usage

```
rasterize_layers(
  plot,
  dpi = 300,
  layers = c("Point", "Jitter", "Line", "Segment", "Linerange", "EdgeSegment")
)
```

Arguments

`plot`	A ggplot object.
`dpi`	Integer DPI for rasterization (default 300).
`layers`	Character vector of layer types to rasterize, e.g. c("Point", "Jitter", "Line", "Segment").

Value

ggplot object with target layers rasterized.

Examples

```
library(ggplot2); library(ggrastr)
plot <- ggplot(diamonds, aes(carat, price, colour = cut)) + geom_point()
rasterize_layers(plot, dpi = 100)
```

recode_by_map *Recode a Vector Using a Two-Column Mapping Table*

Description

This function takes a vector `x` and a mapping data frame `map_df` that has two columns: one with the original values and one with the replacement values. Internally it builds a named vector via `tibble::deframe()`, then applies `dplyr::recode()` with splicing (`!!!`) to perform the recoding.

Usage

```
recode_by_map(x, map_df, from, to)
```

Arguments

`x`	A vector whose elements you want to recode.
`map_df`	A data frame or tibble containing exactly two columns: one for the old values, one for the new values.
`from`	A string giving the name of the column in `map_df` that contains the original values.
`to`	A string giving the name of the column in `map_df` that contains the replacement values.

Details

Build a 2-col mapping df, use deframe() then recode(col, !!!name_vector).

Value

A vector (same type as `x`) with all values matching `map_df[[from]]` replaced by the corresponding `map_df[[to]]`; all other values in `x` are left unchanged.

See Also

recode, deframe

Examples

```
library(dplyr)
map_df <- tibble::tibble(
  old = c("ENSG1", "ENSG2"),
  new = c("TP53",  "BRCA1")
)
recode_by_map(c("ENSG1","foo","ENSG2"), map_df, "old", "new")
#> [1] "TP53" "foo"  "BRCA1"
```

set_proxy *Set or unset proxy*

Description

Quickly enable or disable HTTP/HTTPS (and optionally SOCKS5) proxy for the current R session.

Usage

```
set_proxy(enable = TRUE, host = "127.0.0.1", port = 7897, socks = TRUE)
```

Arguments

enable	TRUE to enable, FALSE to disable.
host	Proxy host address.
port	Proxy port number.
socks	Whether to also set SOCKS5 proxy.

Examples

```
# Enable proxy (HTTP/HTTPS + SOCKS5)
set_proxy(TRUE, host = "127.0.0.1", port = 7897, socks = TRUE)

# Test proxy
system("curl cip.cc")

# Disable proxy
set_proxy(FALSE)
```

vd *View DataFrame with VisiData from Terminal*

Description

`vd()` writes `df` to a temporary TSV file, then opens it with VisiData. This keeps the R side simple while letting VisiData read the table lazily from disk, which is much more usable for large data frames than printing them in the console. Non-default row names are preserved as the first column. A compact terminal summary is printed before launch, including dimensions, row-name handling, temp-file size, and writer backend.

Usage

```
vd(df)
```

Arguments

df	Data frame, tibble, or matrix to view.

Value

Invisibly returns df.

Examples

```
## Not run:
data(mtcars)
vd(mtcars)
mtcars %>% vd()

## End(Not run)
```

Index