Package 'thisutils'

August 19, 2025

Type Package

Title Collection of Utility Functions for Data Analysis and Computing

Version 0.1.2

Date 2025-08-19

Maintainer Meng Xu <mengxu98@qq.com>

Description Provides utility functions for data analysis and scientific computing. Includes functions for parallel processing, and other computational tasks to streamline workflows.

License MIT + file LICENSE

URL https://mengxu98.github.io/thisutils/

BugReports https://github.com/mengxu98/thisutils/issues

Depends R (>= 4.1.0)

Imports cli, doParallel, foreach, Matrix, methods, parallel, purrr, Rcpp, RcppArmadillo, RcppParallel, rlang, stats

Suggests httr2

LinkingTo Rcpp, RcppArmadillo, RcppParallel

Config/Needs/website mengxu98/thistemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

NeedsCompilation yes

Author Meng Xu [aut, cre] (ORCID: <https://orcid.org/0000-0002-8300-1054>)

Repository CRAN

Date/Publication 2025-08-19 07:50:02 UTC

Contents

`thisutils-package` *Collection of Utility Functions for Data Analysis and Computing*

Description

Provides utility functions for data analysis and scientific computing. Includes functions for parallel processing, and other computational tasks to streamline workflows.

Author(s)

Meng Xu (Maintainer), <mengxu98@qq.com>

Source

https://mengxu98.github.io/thisutils/

See Also

Useful links:

- https://mengxu98.github.io/thisutils/
- Report bugs at https://github.com/mengxu98/thisutils/issues

---

`add_pkg_file` *Add package file*

Description

Automatically generate a file containing functions and related code for R package development.

Usage

```
add_pkg_file(
  desc_file,
  pkg_name = NULL,
  title = NULL,
  pkg_description = NULL,
  author_name = NULL,
  author_email = NULL,
  github_url = NULL,
  output_dir = NULL,
  use_figlet = TRUE,
  figlet_font = "Slant",
 colors = c("red", "yellow", "green", "magenta", "cyan", "yellow", "green", "white",
    "magenta", "cyan"),
  unicode = TRUE,
  verbose = TRUE
)
```

Arguments

`desc_file`	The DESCRIPTION file. Must be provided, it will be used to extract package information. Using `add_pkg_file("DESCRIPTION", output_dir = "R")`, will be created `<pkg_name>-package.R` based on the DESCRIPTION file in `"R"` directory. If you want to use some specific information, such as `author_name` or `author_email`, you can provide them manually.
`pkg_name`	Character string, the name of the package. Default is `NULL`, which will be read from DESCRIPTION file.
`title`	Character string, title of the package. Default is `NULL`, which will be read from DESCRIPTION file.
`pkg_description`	Character string, short description of the package. Default is `NULL`, which will be read from DESCRIPTION file.

author_name	Character string, name of the package author. Default is NULL, which will be read from DESCRIPTION file.
author_email	Character string, email of the package author. Default is NULL, which will be read from DESCRIPTION file.
github_url	Character string, GitHub URL of the package. Default is NULL, which will be read from DESCRIPTION file or constructed based on package name.
output_dir	Character string, directory where to save the package file. Default is NULL, you should specify it, such as "R".
use_figlet	Whether to use figlet for ASCII art generation. Default is TRUE. Details see figlet.
figlet_font	Character string, figlet font to use. Default is "Slant".
colors	Character vector, colors to use for the logo elements.
unicode	Whether to use Unicode symbols. Default is TRUE.
verbose	Whether to print progress messages. Default is TRUE.

Value

Creates a file in specified output directory.

---

as_matrix *Convert matrix into dense/sparse matrix*

Description

Convert matrix into dense/sparse matrix

Usage

```
as_matrix(x, return_sparse = FALSE)
```

Arguments

x	A matrix.
return_sparse	Whether to output a sparse matrix. Default is FALSE.

Value

A dense or sparse matrix.

Examples

```
m <- simulate_sparse_matrix(
  1000, 1000,
  decimal = 3
)

a <- as_matrix(m)
a[1:5, 1:5]

b <- as_matrix(m, return_sparse = TRUE)
b[1:5, 1:5]
```

---

capitalize *Capitalize the first letter of each word*

Description

Capitalizes the characters making the first letter uppercase.

Usage

```
capitalize(x, force_tolower = FALSE)
```

Arguments

x	A vector of character strings to be capitalized.
force_tolower	Whether to force the remaining letters to be lowercase.

Examples

```
x <- c(
  "hello world",
  "Hello world",
  "hello World"
)
capitalize(x)
```

check_sparsity *Check sparsity of matrix*

Description

Check sparsity of matrix

Usage

```
check_sparsity(x)
```

Arguments

x	A matrix.

Value

Sparsity of matrix.

download *Download file from the Internet*

Description

Download file from the Internet

Usage

```
download(
  url,
  destfile,
  methods = c("auto", "wget", "libcurl", "curl", "wininet", "internal"),
  quiet = FALSE,
  ...,
  max_tries = 2
)
```

Arguments

url	a character string (or longer vector for the "libcurl" method) naming the URL of a resource to be downloaded.
destfile	a character string (or vector, see the url argument) with the file path where the downloaded file is to be saved. Tilde-expansion is performed.
methods	Methods to be used for downloading files. Default is "auto", which means to try different download methods when the current method fails.

quiet	If TRUE, suppress status messages (if any), and the progress bar.
...	Other arguments passed to utils::download.file.
max_tries	Number of tries for each download method. Default is 2.

---

figlet *The figlet function*

Description

Create ASCII art text using figlet.

Usage

```
figlet(
  text,
  font = "Slant",
  width = getOption("width", 80),
  justify = "left",
  absolute = FALSE,
  strip = TRUE
)
```

Arguments

text	Text to make bigger.
font	Name of font, path to font, or figlet_font object.
width	Width to use when justifying and breaking lines.
justify	Text justification to use in rendering ("left", "centre", "right").
absolute	Logical, indicating if alignment is absolute.
strip	Logical, indicating if whitespace should be removed.

Value

An object of class figlet_text which is a character vector with a handy print method.

References

https://github.com/richfitz/rfiglet, https://github.com/jbkunst/figletr, http://www.figlet.org/

Examples

```
figlet("thisutils")
```

figlet_font *Get a figlet font*

Description

Get a figlet font

Usage

```
figlet_font(font)
```

Arguments

font	Path or name of the font to load

Value

A ‘figlet_font‘ object for use with [figlet]

invoke_fun *Invoke a function with a list of arguments*

Description

Invoke a function with a list of arguments

Usage

```
invoke_fun(.fn, .args = list(), ..., .env = rlang::caller_env())
```

Arguments

.fn	A function, or function name as a string.
.args	A list of arguments.
...	Other arguments passed to the function.
.env	Environment in which to evaluate the call. This will be most useful if .fn is a string, or the function has side-effects.

Examples

```
f <- function(x, y) {
  x + y
}
invoke_fun(f, list(x = 1, y = 2))
invoke_fun("f", list(x = 1, y = 2))
invoke_fun("f", x = 1, y = 2)
```

list_figlet_fonts *List available figlet fonts*

Description

List all figlet font files available in the package or system.

Usage

```
list_figlet_fonts()
```

Value

Character vector of available font names.

Examples

```
list_figlet_fonts()
```

log_message *Print formatted message*

Description

Integrate the message printing function with the `cli` package, and the base::message function. The message could be suppressed by base::suppressMessages.

Usage

```
log_message(
  ...,
  verbose = TRUE,
  message_type = c("info", "success", "warning", "error"),
  cli_model = TRUE,
  level = 1,
  symbol = "  ",
  text_color = NULL,
  back_color = NULL,
  text_style = NULL,
  multiline_indent = FALSE,
  timestamp = TRUE,
  timestamp_format = paste0("[", format(Sys.time(), "%Y-%m-%d %H:%M:%S"), "] "),
  timestamp_style = TRUE,
  .envir = parent.frame(),
  .frame = .envir
)
```

Arguments

`...`	The message to print.
`verbose`	Whether to print the message. Default is `TRUE`.
`message_type`	Type of message. Could be choose one of `info`, `success`, `warning`, and `error`. Default is `info`.
`cli_model`	Whether to use the `cli` package to print the message. Default is `TRUE`.
`level`	The level of the message, which affects the indentation. Level 1 has no indentation, higher levels add more indentation. Default is `1`.
`symbol`	The symbol used for indentation. When specified, it ignores the level parameter and uses the symbol directly. Default is `" "` (two spaces).
`text_color`	Color for the message text. Supports R color names (e.g., "orange"), hexadecimal colors (e.g., "#000000"), basic colors: "red", "green", "blue", "yellow", "magenta", "cyan", "white", "black", "grey", "silver", "none", and bright colors: "br_red", "br_green", "br_blue", "br_yellow", "br_magenta", "br_cyan", "br_white", "br_black". Default is `NULL`.
`back_color`	Background color for the message text. Details see parameter `text_color`. Default is `NULL`.
`text_style`	Text styles to apply. Can be one or more of: "bold", "italic", "underline", "strikethrough", "dim", "inverse". Multiple styles can be combined (e.g., c("bold", "underline")). Default is `NULL`.
`multiline_indent`	Whether to apply consistent formatting (timestamp and indentation) to each line in multiline messages. When TRUE, each line gets the full formatting; when FALSE, only the first line gets the timestamp. Default is `FALSE`.
`timestamp`	Whether to show the current time in the message. Default is `TRUE`.
`timestamp_format`	Format string for timestamp display. Default is `"%Y-%m-%d %H:%M:%S"`.
`timestamp_style`	Whether to apply the same text styling to the timestamp as the message text. When TRUE, timestamp formatting matches the message; when FALSE, timestamp keeps its default appearance. Default is `TRUE`.
`.envir`	The environment to evaluate calls in. Default is `parent.frame()`.
`.frame`	The frame to use for error reporting. Default is `.envir`.

Value

`Formated message.`

References

https://cli.r-lib.org/articles/index.html

Examples

```
# basic usage
log_message("Hello, ", "world!")

log_message("hello, world!")

log_message("Hello, world!", timestamp = FALSE)

log_message(
  "Hello, ", "world!",
  message_type = "success"
)

log_message(
  "Hello, world!",
  message_type = "warning"
)

log_message(
  "Hello, ", "world!",
  cli_model = FALSE
)


# suppress messages
suppressMessages(log_message("Hello, world!"))
log_message("Hello, world!", verbose = FALSE)
options(log_message.verbose = FALSE)
log_message("Hello, world!")

# for global verbose option
options(log_message.verbose = TRUE)
log_message("Hello, world!", verbose = FALSE)
options(log_message.verbose = NULL)


# cli inline markup
log_message("{.arg abc} is a argument")

## 'message' can not deal with cli inline markup
message("hello, {.code world}!")

log_message("{.val list('abc')} is a {.cls {class(list('abc'))}}")

log_message("{.code lm(y ~ x)} is a code example")

log_message("{.dt List}list('abc')")

log_message("address: {.email example@example.com}")

log_message("{.emph R} is a programming language")
```

```
log_message("{.envvar R_HOME}")

log_message("{.file log_message.R} is a file")

log_message("{.fn lm} is a function")

log_message("{.fun lm} is a function")

log_message("{.help lm} to get help")

log_message("... see {.help [{.fun lm}](stats::lm)} to learn more")

log_message(
  "See the {.href [cli homepage](https://cli.r-lib.org)} for details"
)

log_message("press {.kbd ENTER}")

log_message("press {.key ENTER}")

log_message("URL: {.url https://cli.r-lib.org}")

log_message("Some {.field field}")

log_message("{.path /usr/bin/R} is a path")

log_message("{.pkg cli} is a package")

log_message("{.val object} is a variable")

log_message("{.run Rscript log_message.R} is a runnable file")

log_message("{.str object} is a string")

log_message("{.strong abc} is a strong string")

log_message("{.topic stats::lm} is a topic")

log_message("{.vignette cli} is a vignette")


# set indentation
log_message("Hello, world!", level = 2)

log_message("Hello, world!", symbol = "->")

log_message(
  "Hello, world!",
  symbol = "#####",
  level = 3
)

# color formatting
```

```
log_message(
  "This is a red message",
  text_color = "#ff9900"
)

log_message(
  "This is a message with background",
  back_color = "#EE4000"
)

log_message(
  "This is a message with both text and background",
  text_color = "white",
  back_color = "cyan"
)

log_message(
  "This is a message with background",
  back_color = "#EE4000",
  cli_model = FALSE
)

log_message(
  "This is a message with both text and background",
  text_color = "red",
  back_color = "cyan",
  cli_model = FALSE
)

log_message(
  "Hex color with {.arg cli_model = FALSE}",
  text_color = "#FF5733",
  cli_model = FALSE
)

log_message(
  "Bright red text",
  text_color = "br_red"
)

log_message(
  "Bright background",
  back_color = "br_yellow"
)

log_message(
  "Combined grey and style",
  text_color = "grey",
  text_style = "bold"
)

# text style formatting
log_message(
```

```
  "Bold message",
  text_style = "bold"
)

log_message(
  "Italic message",
  text_style = "italic"
)

log_message(
  "Underlined message",
  text_style = "underline"
)

log_message(
  "Combined styles",
  text_style = c("bold", "underline")
)

log_message(
  "Color and style",
  text_color = "blue",
  text_style = c("bold", "italic")
)

log_message(
  "Hex color and style",
  text_color = "#FF5733",
  text_style = c("bold", "underline")
)


# multiline message
log_message(
  "Line 1\nLine 2\nLine 3",
  multiline_indent = TRUE,
  text_style = "italic"
)

log_message(
  "Multi-line\ncolored\nmessage",
  text_color = "blue",
  text_style = "italic"
)

log_message(
  "Multi-line\ncolored\nmessage",
  text_color = "blue",
  timestamp = FALSE
)

# timestamp styling
log_message(
```

```
  "Multi-line message\nwith timestamp styling",
  text_color = "red",
  text_style = "bold",
  timestamp_style = TRUE
)

log_message(
  "Multi-line message\nwithout timestamp styling",
  text_color = "#669999",
  text_style = c("bold", "italic"),
  timestamp_style = FALSE
)


# combine cli package and log_message
log_message(
  cli::col_green(
    "I am a green line ",
    cli::col_blue(
      cli::style_underline(
        cli::style_bold("with a blue substring")
      )
    ),
    " that becomes green again!"
  )
)

# cli variables
fun <- function(x = 1) {
  log_message("{.val x}")
  log_message("{.val {x}}")
  log_message("{.val {x + 1}}")
}
fun()
```

`matrix_process` *Process matrix*

Description

Process matrix

Usage

```
matrix_process(
  matrix,
  method = c("raw", "zscore", "fc", "log2fc", "log1p"),
  ...
)
```

Arguments

matrix	A matrix.
method	Method to use for processing the matrix.
...	Other arguments passed to the method.

Value

A processed matrix.

Examples

```
m <- simulate_sparse_matrix(10, 10)
matrix_process(m, method = "raw")
matrix_process(m, method = "zscore")
matrix_process(m, method = "fc")
matrix_process(m, method = "log2fc")
matrix_process(m, method = "log1p")
m <- as_matrix(m)
matrix_process(m, method = function(x) x / rowMeans(x))
```

---

matrix_to_table *Switch matrix to table*

Description

Switch matrix to table

Usage

```
matrix_to_table(
  matrix,
  row_names = NULL,
  col_names = NULL,
  threshold = 0,
  keep_zero = FALSE
)
```

Arguments

matrix	A matrix.
row_names	Character vector of row names to filter by.
col_names	Character vector of column names to filter by.
threshold	The threshold for filtering values based on absolute values. Defaults to 0.
keep_zero	Whether to keep zero values in the table. Defaults to false.

Value

A table with three columns: `row`, `col`, and `value`.

Examples

```
test_matrix <- simulate_sparse_matrix(10, 10)
colnames(test_matrix) <- paste0("c", 1:10)
rownames(test_matrix) <- paste0("r", 1:10)
table <- matrix_to_table(test_matrix)
matrix_new <- table_to_matrix(table)
test_matrix <- test_matrix[rownames(matrix_new), colnames(matrix_new)] |>
  as_matrix()
identical(test_matrix, matrix_new)

matrix_to_table(
  test_matrix,
  threshold = 2
)

matrix_to_table(
  test_matrix,
  row_names = c("r1", "r2"),
  col_names = c("c1", "c2")
)
```

`normalization` *Normalize numeric vector*

Description

Normalize numeric vector

Usage

```
normalization(x, method = "max_min", na_rm = TRUE, ...)
```

Arguments

`x`	Input numeric vector.
`method`	Method used for normalization.
`na_rm`	Whether to remove `NA` values, and if setting `TRUE`, using `0` instead. Default is `TRUE`.
`...`	Parameters for other methods.

Value

Normalized numeric vector.

Examples

```
x <- c(runif(2), NA, -runif(2))
x
normalization(x, method = "max_min")
normalization(x, method = "maximum")
normalization(x, method = "sum")
normalization(x, method = "softmax")
normalization(x, method = "z_score")
normalization(x, method = "mad")
normalization(x, method = "unit_vector")
normalization(x, method = "unit_vector", na_rm = FALSE)
```

---

parallelize_fun *Parallelize a function*

Description

Parallelize a function

Usage

```
parallelize_fun(x, fun, cores = 1, export_fun = NULL, verbose = TRUE)
```

Arguments

x	A vector or list to apply over.
fun	The function to be applied to each element.
cores	The number of cores to use for parallelization with foreach::foreach. Default is 1.
export_fun	The functions to export the function to workers.
verbose	Whether to print progress messages. Default is TRUE.

Value

A list of computed results.

Examples

```
parallelize_fun(1:3, function(x) {
  Sys.sleep(0.2)
  x^2
})

parallelize_fun(list(1, 2, 3), function(x) {
  Sys.sleep(0.2)
  x^2
}, cores = 2)
```

pearson_correlation *Correlation and covariance calculation for sparse matrix*

Description

Correlation and covariance calculation for sparse matrix

Usage

```
pearson_correlation(x, y = NULL)
```

Arguments

x	Sparse matrix or character vector.
y	Sparse matrix or character vector.

Value

A list with covariance and correlation matrices.

Examples

```
m1 <- simulate_sparse_matrix(
  100, 100
)
m2 <- simulate_sparse_matrix(
  100, 100,
  sparsity = 0.05
)
a <- pearson_correlation(m1, m2)
a$cov[1:5, 1:5]
a$cor[1:5, 1:5]
```

print.thisutils_logo *Print logo*

Description

Print logo

Usage

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

Arguments

x	Input information.
...	Other parameters.

Value

Print the ASCII logo

---

remove_space *Remove and normalize spaces*

Description

Remove leading/trailing spaces and normalize multiple spaces between words in character strings.

Usage

```
remove_space(
  x,
  trim_start = TRUE,
  trim_end = FALSE,
  collapse_multiple = TRUE,
  preserve_newlines = TRUE
)
```

Arguments

x	A vector of character strings.
trim_start	Logical value, default is TRUE. Whether to remove leading spaces before the first word.
trim_end	Logical value, default is FALSE. Whether to remove trailing spaces after the last word.
collapse_multiple	Logical value, default is TRUE. Whether to collapse multiple consecutive spaces between words into a single space.
preserve_newlines	Logical value, default is TRUE. Whether to preserve newline characters when collapsing spaces.

Value

A character vector with spaces normalized according to the specified parameters.

Examples

```
x <- c(
  " hello  world ",
  "  test   case  ",
  "no space",
  "   multiple   spaces   "
)
remove_space(x)
remove_space(x, trim_start = FALSE)
remove_space(x, trim_end = TRUE)
remove_space(x, collapse_multiple = FALSE)
remove_space(
  x,
  trim_start = FALSE,
  trim_end = FALSE,
  collapse_multiple = FALSE
)

# with newlines
multiline <- c(
  "hello\n\n  world  ",
  "  first  \n  second  "
)
remove_space(multiline)
remove_space(multiline, preserve_newlines = FALSE)
```

---

rescale *Rescale numeric vector*

Description

Rescale numeric vector

Usage

```
rescale(x, from = range(x, na.rm = TRUE, finite = TRUE), to = c(0, 1))
```

Arguments

x	A numeric vector.
from	The range of the original data.
to	The range of the rescaled data.

Value

A numeric vector with rescaled values.

Examples

```
x <- rnorm(10)
rescale(x)
rescale(x, from = c(0, 1))
rescale(x, to = c(0, 2))
```

`r_square` *Coefficient of determination (R^2)*

Description

Coefficient of determination (R^2)

Usage

```
r_square(y_true, y_pred)
```

Arguments

`y_true`	A numeric vector with ground truth values.
`y_pred`	A numeric vector with predicted values.

Value

The R^2 value.

Examples

```
y <- rnorm(100)
y_pred <- y + rnorm(100, sd = 0.5)
r_square(y, y_pred)
```

`simulate_sparse_matrix`
Generate a simulated sparse matrix

Description

This function generates a sparse matrix with a specified number of rows and columns, a given sparsity level, and a distribution function for the non-zero values.

Usage

```
simulate_sparse_matrix(
  nrow,
  ncol,
  sparsity = 0.95,
  distribution_fun = function(n) stats::rpois(n, lambda = 0.5) + 1,
  decimal = 0,
  seed = 1
)
```

Arguments

nrow	Number of rows in the matrix.
ncol	Number of columns in the matrix.
sparsity	Proportion of zero elements (sparsity level). Default is 0.95, meaning 95% of elements are zero (5% are non-zero).
distribution_fun	Function to generate non-zero values.
decimal	Controls the number of decimal places in the generated values. If set to 0, values will be integers. When decimal > 0, random decimal parts are uniformly distributed across the full range. Default is 0.
seed	Random seed for reproducibility.

Value

A sparse matrix of class "dgCMatrix".

Examples

```
simulate_sparse_matrix(1000, 500) |>
  check_sparsity()

simulate_sparse_matrix(10, 10, decimal = 1)
simulate_sparse_matrix(10, 10, decimal = 5)
```

---

sparse_cor *Sparse correlation function*

Description

Safe correlation function which returns a sparse matrix without missing values.

Usage

```
sparse_cor(
  x,
  y = NULL,
  method = "pearson",
  allow_neg = TRUE,
  remove_na = TRUE,
  remove_inf = TRUE,
  ...
)
```

Arguments

x	Sparse matrix or character vector.
y	Sparse matrix or character vector.
method	Method to use for calculating the correlation coefficient.
allow_neg	Logical. Whether to allow negative values or set them to 0.
remove_na	Logical. Whether to replace NA values with 0.
remove_inf	Logical. Whether to replace infinite values with 1.
...	Other arguments passed to stats::cor function.

Value

A correlation matrix.

Examples

```
m1 <- simulate_sparse_matrix(
  500, 100
)
m2 <- simulate_sparse_matrix(
  500, 100,
  seed = 2025
)
a <- sparse_cor(m1)
b <- sparse_cor(m1, m2)
c <- as_matrix(
  cor(as_matrix(m1)),
  return_sparse = TRUE
)
d <- as_matrix(
  cor(as_matrix(m1), as_matrix(m2)),
  return_sparse = TRUE
)

a[1:5, 1:5]
c[1:5, 1:5]
all.equal(a, c)
```

```
b[1:5, 1:5]
d[1:5, 1:5]
all.equal(b, d)

m1[sample(1:500, 10)] <- NA
m2[sample(1:500, 10)] <- NA

sparse_cor(m1, m2)[1:5, 1:5]

system.time(
  sparse_cor(m1)
)
system.time(
  cor(as_matrix(m1))
)

system.time(
  sparse_cor(m1, m2)
)
system.time(
  cor(as_matrix(m1), as_matrix(m2))
)
```

`split_indices` *Split indices.*

Description

An optimised version of split for the special case of splitting row indices into groups.

Usage

```
split_indices(group, n = 0L)
```

Arguments

`group`	Integer indices
`n`	The largest integer (may not appear in index). This is hint: if the largest value of `group` is bigger than `n`, the output will silently expand.

Value

A list of vectors of indices.

References

https://github.com/hadley/plyr/blob/d57f9377eb5d56107ba3136775f2f0f005f33aa3/src/split-numeric.cpp#L20

Examples

```
split_indices(sample(10, 100, rep = TRUE))
split_indices(sample(10, 100, rep = TRUE), 10)
```

`str_wrap` *Wrap text*

Description

Wrap text

Usage

```
str_wrap(x, width = 80)
```

Arguments

`x`	A character vector to wrap.
`width`	The maximum width of the lines.

Value

A character vector with wrapped text.

Examples

```
str_wrap(rep("Hello, world!", 10))
str_wrap(rep("Hello, world!", 10), width = 10)
```

`table_to_matrix` *Switch table to matrix*

Description

Switch table to matrix

Usage

```
table_to_matrix(
  table,
  row_names = NULL,
  col_names = NULL,
  threshold = 0,
  return_sparse = FALSE
)
```

Arguments

`table`	A table with three columns: `row`, `col`, and `value`.
`row_names`	Character vector of row names to filter by.
`col_names`	Character vector of column names to filter by.
`threshold`	The threshold for filtering values based on absolute values. Defaults to `0`.
`return_sparse`	Whether to return a sparse matrix. Defaults to `false`.

Value

A matrix.

Examples

```
table <- data.frame(
  row = c("r1", "r2", "r3", "r4", "r5", "r6"),
  col = c("c4", "c5", "c6", "c1", "c2", "c3"),
  value = c(0.6, -0.5, -0.4, 0.3, 0.2, 0.1)
)
matrix <- table_to_matrix(table)
table_new <- matrix_to_table(matrix)
identical(table, table_new)

table_to_matrix(table, threshold = 0.3)

table_to_matrix(
  table,
  row_names = c("r1", "r2"),
  col_names = c("c4", "c5")
)

sparse_matrix <- simulate_sparse_matrix(10, 10)
table_sparse <- matrix_to_table(
  sparse_matrix,
  keep_zero = TRUE
)
sparse_matrix_new <- table_to_matrix(
  table_sparse,
  return_sparse = TRUE
)
identical(sparse_matrix, sparse_matrix_new)
```

`thisutils_logo` *The logo of thisutils*

Description

The thisutils logo, using ASCII or Unicode characters Use cli::ansi_strip to get rid of the colors.

Usage

```
thisutils_logo(unicode = cli::is_utf8_output())
```

Arguments

unicode	Unicode symbols on UTF-8 platforms. Default is cli::is_utf8_output.

References

https://github.com/tidyverse/tidyverse/blob/main/R/logo.R

Examples

```
thisutils_logo()
```

---

try_get *Try to evaluate an expression a set number of times before failing*

Description

The function is used as a fail-safe if your R code sometimes works and sometimes doesn't, usually because it depends on a resource that may be temporarily unavailable. It tries to evaluate the expression `max_tries` times. If all the attempts fail, it throws an error; if not, the evaluated expression is returned.

Usage

```
try_get(expr, max_tries = 5, error_message = "", retry_message = "Retrying...")
```

Arguments

expr	The expression to be evaluated.
max_tries	The maximum number of attempts to evaluate the expression before giving up. Default is set to 5.
error_message	Additional custom error message to be displayed when an error occurs.
retry_message	Message displayed when a new try to evaluate the expression would be attempted.

Value

The evaluated expression if successful, otherwise it throws an error if all attempts are unsuccessful.

Examples

```
f <- function() {
  value <- runif(1, min = 0, max = 1)
  if (value > 0.5) {
    log_message("value is larger than 0.5")
    return(value)
  } else {
    log_message(
      "value is smaller than 0.5",
      message_type = "error"
    )
  }
}
f_evaluated <- try_get(expr = f())
print(f_evaluated)
```

---

unnest_fun *Unnest a list-column*

Description

Implement similar functions to the tidyr::unnest function.

Usage

```
unnest_fun(data, cols, keep_empty = FALSE)
```

Arguments

data	A data frame.
cols	Columns to unnest.
keep_empty	By default, you get one row of output for each element of the list your unchopping/unnesting. This means that if there's a size-0 element (like NULL or an empty data frame), that entire row will be dropped from the output. If you want to preserve all rows, use keep_empty = TRUE to replace size-0 elements with a single row of missing values.

Examples

```
data <- data.frame(
  id = 1:3,
  x = c("a", "b", "c"),
  stringsAsFactors = FALSE
)
data$data <- list(
  c(1, 2),
  c(3, 4, 5),
  c(6)
```

```
)
unnest_fun(data, cols = "data")

data2 <- data.frame(
  id = 1:3,
  x = c("a", "b", "c"),
  stringsAsFactors = FALSE
)
data2$data <- list(
  c(1, 2),
  numeric(0),
  c(6)
)
unnest_fun(data2, cols = "data")
unnest_fun(data2, cols = "data", keep_empty = TRUE)
```

`%ss%` *Value selection operator*

Description

This operator returns the left side if it's not NULL, otherwise it returns the right side.

Usage

```
a %ss% b
```

Arguments

a	The left side value to check
b	The right side value to use if a is NULL

Value

a if it is not NULL, otherwise b

Examples

```
NULL %ss% 10
5 %ss% 10
```

Index