

Package: vitopack (via r-universe)

May 12, 2026

Type Package

Title Actuarial Helpers for Triangles, Exposures and Czech Birth Numbers

Version 0.1.1

Description A collection of utilities that grew out of day-to-day non-life actuarial work at Com-PASS Advisory. Provides helpers for building chain-ladder triangles (cumulative, decumulative, run-off, development factors with optional weighting), constructing exposure columns from policy start/end dates, parsing Czech birth numbers ('rodné číslo') into dates, generating smooth RGB color palettes for charts, and loading multi-sheet 'xlsx'/'xlsb' files into a list of data frames. The chain-ladder helpers follow the standard methodology of Mack (1993) <doi:10.2143/AST.23.2.2005092>.

License MIT + file LICENSE

URL <https://github.com/Com-PASS-OV/vitopack>,
<https://Com-PASS-OV.github.io/vitopack/>

BugReports <https://github.com/Com-PASS-OV/vitopack/issues>

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports data.table, dplyr, tibble, tidyselect, rlang, lubridate,
grDevices, graphics, stats, utils

Suggests readxl, plotly, testthat (>= 3.0.0), knitr, rmarkdown, covr

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Ondřej Vít [aut, cre]

Maintainer Ondřej Vít <ondrej.vit@com-pass.cz>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-05-12 21:45:42 UTC

RemoteUrl <https://github.com/cran/vitopack>

RemoteRef HEAD

RemoteSha 80b22969124433bc360e41f78d8f6a91faebc3f3

Contents

bind_with_source	3
create_annual_triangle	3
create_avg_coefs	4
create_chl_coef_triangle	4
create_chl_coefs	5
create_chl_coefs_weighted	6
create_chl_coefs_weighted_future	6
create_chl_trg_visualization	7
create_chl_trg_visualization_pvzp	7
create_cumulative_triangle	8
create_decumulative_triangle	9
create_find_columns	9
create_multiple_columns_m	10
create_policy_exposure_columns_m	11
create_policy_exposure_days_columns	12
create_product_coefs	13
create_run_off_check	14
create_triangle	14
diag_reader	15
diag_sums	16
diag_writer	16
get_colors	17
load_excel_sheets	18
load_xlsb_sheets	18
numeric_format	19
plot_colorBars	20
rainbow_cat	20
rc_to_birth_day	21
rgb_colors_for_plot	22
triangle_namer	22

Index

24

bind_with_source	<i>Row-bind a list of data frames with a source column</i>
------------------	--

Description

Thin wrapper around `data.table::rbindlist()` that prepends a column called `source` with the names of `lst`. Returns `NULL` for an empty input.

Usage

```
bind_with_source(lst)
```

Arguments

`lst` A named list of data frames to bind together.

Value

A `data.table` with all columns from the inputs and an added `source` column, or `NULL` if `lst` is empty.

Examples

```
bind_with_source(list(
  a = data.frame(x = 1:2, y = 3:4),
  b = data.frame(x = 5,   z = 9)
))
```

create_annual_triangle

Re-aggregate a per-period triangle to an annual triangle

Description

Combines period consecutive sub-period diagonals/columns of a cumulative triangle into one annual diagonal/column. Useful when the source data is monthly or quarterly and the report wants annual figures.

Usage

```
create_annual_triangle(cum_trg, period = 4)
```

Arguments

`cum_trg` A cumulative numeric triangle.
`period` Integer - number of sub-periods that make up one year (default 4 for quarters).

Value

A cumulative annual triangle (numeric matrix).

Examples

```
# 4x4 quarterly cumulative triangle (only the lower-left corner is real data)
q_trg <- matrix(NA_real_, 4, 4)
q_trg[1, ] <- c(1, 2, 3, 4)
q_trg[2, 1:3] <- c(2, 4, 5)
q_trg[3, 1:2] <- c(3, 5)
q_trg[4, 1] <- 4
create_annual_triangle(q_trg, period = 4)
```

create_avg_coefs	<i>Average development factors from a CHL factor triangle</i>
------------------	---

Description

Averages the per-cell development factors of a CHL factor triangle along diagonals. For each requested window length, returns one row with the simple-average factor per development period.

Usage

```
create_avg_coefs(chl_trg, avg_length = "full")
```

Arguments

chl_trg	A development-factor triangle (e.g. the output of <code>create_chl_coef_triangle()</code>).
avg_length	Vector – each element either "full" (use the full diagonal) or an integer giving the window length.

Value

A data.frame with one row per avg_length element.

create_chl_coef_triangle	<i>Element-wise chain-ladder development factor triangle</i>
--------------------------	--

Description

For every cell (i, j) of a cumulative triangle returns $\text{cum_trg}[i, j] / \text{cum_trg}[i, j-1]$. The first column and the upper-right NA corner are left as NA.

Usage

```
create_chl_coef_triangle(cum_trg)
```

Arguments

cum_trg A cumulative numeric triangle.

Value

A numeric matrix of the same shape with development factors.

Examples

```
cum <- matrix(c(10, 30, 32, 20, 27, NA, 15, NA, NA), nrow = 3, byrow = TRUE)
create_chl_coef_triangle(cum)
```

create_chl_coefs	<i>Volume-weighted chain-ladder development factors</i>
------------------	---

Description

Computes one row of development factors per requested length ("full" or an integer giving how many of the most recent rows to use). Factors are aggregated using the standard volume-weighted formula $\text{sum}(\text{cum}[, j]) / \text{sum}(\text{cum}[, j - 1])$.

Usage

```
create_chl_coefs(cum_trg, chl_length = "full")
```

Arguments

cum_trg A cumulative numeric triangle.

chl_length A vector – each element is either the literal "full" (use all rows) or a numeric/integer giving the window length. Default "full".

Value

A data.frame with one row per chl_length element, the first column labelling the row and the remaining columns holding the development factors.

Examples

```
cum <- matrix(c(10, 30, 32, 20, 27, NA, 15, NA, NA), nrow = 3, byrow = TRUE)
create_chl_coefs(cum, chl_length = c("full", 2))
```

create_chl_coefs_weighted

Volume-weighted CHL factors with an explicit weight triangle

Description

Same idea as [create_chl_coefs\(\)](#) but every cell is weighted by an arbitrary `trg_weight` matrix evaluated at the **previous** column ($j - 1$). Useful when the natural exposure differs from the cumulative triangle itself (e.g. paid claims weighted by exposure-as-of-prior-period).

Usage

```
create_chl_coefs_weighted(cum_trg, trg_weight, chl_length = "full")
```

Arguments

<code>cum_trg</code>	Cumulative triangle.
<code>trg_weight</code>	Weight matrix of the same shape as <code>cum_trg</code> .
<code>chl_length</code>	As in create_chl_coefs() .

Value

A data.frame analogous to the output of [create_chl_coefs\(\)](#).

create_chl_coefs_weighted_future

Volume-weighted CHL factors with weight at the current column

Description

Variant of [create_chl_coefs_weighted\(\)](#) that uses `trg_weight[, j]` instead of `trg_weight[, j - 1]`, i.e. the weight evaluated at the developed (forward-looking) column.

Usage

```
create_chl_coefs_weighted_future(cum_trg, trg_weight, chl_length = "full")
```

Arguments

<code>cum_trg</code>	Cumulative triangle.
<code>trg_weight</code>	Weight matrix of the same shape as <code>cum_trg</code> .
<code>chl_length</code>	As in create_chl_coefs() .

Value

A data.frame analogous to the output of [create_chl_coefs\(\)](#).

`create_chl_trg_visualization`*Plotly visualisation of selected columns of a CHL triangle*

Description

Creates an interactive plotly line+marker chart of the selected columns of `data_matrix`, with the row names used as the time axis.

Usage

```
create_chl_trg_visualization(data_matrix, columns)
```

Arguments

`data_matrix` A named matrix where row names act as the x-axis.
`columns` Character or numeric vector – which columns to plot.

Value

A plotly htmlwidget.

Examples

```
## Not run:  
trg <- matrix(c(1, 1.2, 1.05, 1, 1.3, NA, 1, NA, NA), 3,  
             dimnames = list(c("2022", "2023", "2024"), c("0", "1", "2")))  
create_chl_trg_visualization(trg, columns = c("1", "2"))  
  
## End(Not run)
```

`create_chl_trg_visualization_pvzp`*PVZP-style plotly visualisation (string x-axis)*

Description

Like `create_chl_trg_visualization()`, but the x-axis is left as the raw character row names (no coercion to numeric).

Usage

```
create_chl_trg_visualization_pvzp(data_matrix, columns)
```

Arguments

`data_matrix` A named matrix where row names act as the x-axis.
`columns` Character or numeric vector – which columns to plot.

Value

A plotly htmlwidget.

`create_cumulative_triangle`
Cumulate a development triangle along columns

Description

Turns an incremental triangle into a cumulative one by running cumulative sums along the rows (development direction).

Usage

```
create_cumulative_triangle(trg)
```

Arguments

`trg` An incremental triangle (numeric matrix) where the upper-right NA corner is preserved.

Value

A numeric matrix of the same shape, cumulatively summed.

Examples

```
trg <- matrix(c(10, 5, 2, 20, 7, NA, 15, NA, NA), nrow = 3, byrow = TRUE)  
create_cumulative_triangle(trg)
```

```
create_decumulative_triangle
```

De-cumulate a cumulative development triangle

Description

Inverse of `create_cumulative_triangle()`.

Usage

```
create_decumulative_triangle(trg)
```

Arguments

`trg` A cumulative triangle.

Value

The incremental triangle.

Examples

```
cum_trg <- matrix(c(10, 30, 32, 20, 27, NA, 15, NA, NA), nrow = 3, byrow = TRUE)
create_decumulative_triangle(cum_trg)
```

```
create_find_columns    Add zero/value columns based on equality conditions
```

Description

For each pair (`condition_names[i]`, `new_cols_names[i]`) the function adds a new column to data initialised to 0. Where the row matches `data[[condition_var]] == condition_names[i]`, the new column is set to the value of `data[[values_var]]` (evaluated as an expression - see Details).

Usage

```
create_find_columns(
  data,
  condition_names,
  new_cols_names,
  condition_var,
  values_var
)
```

Arguments

data	A data.frame or data.table.
condition_names	Character vector of values to match against condition_var.
new_cols_names	Character vector with the names of the new columns to create. Same length as condition_names.
condition_var	Character scalar - name of the column to test.
values_var	Character scalar - expression that yields the value to assign where the condition matches.

Details

The values_var argument is parsed and evaluated inside data[, ...], which means it can be a bare column name or a small data.table-style expression. The function modifies data by reference (via `data.table::setDT()`) and also returns it invisibly to allow chaining.

Value

The updated data.table (modified in place).

Examples

```
dt <- data.frame(line = c("MTPL", "CASCO", "MTPL"), premium = c(100, 200, 300))
create_find_columns(
  data = dt,
  condition_names = c("MTPL", "CASCO"),
  new_cols_names = c("MTPL_premium", "CASCO_premium"),
  condition_var = "line",
  values_var = "premium"
)
```

```
create_multiple_columns_m
```

Multiply a set of columns by another column

Description

For each multiple_exp_names[i] the function creates a new column new_variable_names[i] equal to multiple_exp_names[i] * multiply_var.

Usage

```
create_multiple_columns_m(
  data,
  new_variable_names,
  multiple_exp_names,
  multiply_var
)
```

Arguments

data A data frame.
new_variable_names Character vector - names of the resulting columns.
multiple_exp_names Character vector - names of the columns to multiply.
multiply_var Character scalar - name of the multiplier column.

Value

A data frame with the new multiplied columns added.

Examples

```

df <- data.frame(exp_q1 = c(0.25, 0.5), exp_q2 = c(0.5, 0.5), premium = c(100, 200))
create_multiple_columns_m(
  data = df,
  new_variable_names = c("prem_q1", "prem_q2"),
  multiple_exp_names = c("exp_q1", "exp_q2"),
  multiply_var = "premium"
)

```

```
create_policy_exposure_columns_m
```

Build per-period exposure columns (in years)

Description

For each requested exposure period [start_months[i], end_months[i]] the function adds a column exp_names[i] to data containing the overlap of that period with each row's policy span [start_policy_var, end_policy_var], expressed in **years** (days / 365).

Usage

```

create_policy_exposure_columns_m(
  data,
  exp_names,
  start_months,
  end_months,
  start_policy_var,
  end_policy_var
)

```

Arguments

data A data.frame or data.table.
exp_names Character vector with the names of the new exposure columns.
start_months, end_months Character vectors of period boundaries that can be coerced via `base::as.Date()`. Same length as `exp_names`.
start_policy_var, end_policy_var Names of the columns holding the policy start and end dates.

Value

The updated data.table (modified in place).

Examples

```

dt <- data.frame(
  policy_start = as.Date(c("2024-01-15", "2024-06-01")),
  policy_end   = as.Date(c("2024-12-31", "2025-05-31"))
)
create_policy_exposure_columns_m(
  data           = dt,
  exp_names      = c("exp_2024_Q1", "exp_2024_Q2"),
  start_months   = c("2024-01-01", "2024-04-01"),
  end_months     = c("2024-03-31", "2024-06-30"),
  start_policy_var = "policy_start",
  end_policy_var  = "policy_end"
)

```

create_policy_exposure_days_columns
Build per-period exposure columns (in days)

Description

Sister of `create_policy_exposure_columns_m()` - same logic, but the exposure is expressed in **days** (rounded) instead of years.

Usage

```

create_policy_exposure_days_columns(
  data,
  exp_names,
  start_months,
  end_months,
  start_policy_var,
  end_policy_var
)

```

Arguments

data A data.frame or data.table.
exp_names Character vector with the names of the new exposure columns.
start_months, end_months Character vectors of period boundaries that can be coerced via `base::as.Date()`. Same length as `exp_names`.
start_policy_var, end_policy_var Names of the columns holding the policy start and end dates.

Value

The updated data.table (modified in place).

Examples

```

dt <- data.frame(
  policy_start = as.Date("2024-01-15"),
  policy_end   = as.Date("2024-12-31")
)
create_policy_exposure_days_columns(
  data           = dt,
  exp_names      = "exp_days_q1",
  start_months   = "2024-01-01",
  end_months     = "2024-03-31",
  start_policy_var = "policy_start",
  end_policy_var  = "policy_end"
)

```

create_product_coefs *Cumulative product of chain-ladder factors (ultimate development)*

Description

Given a one-row data frame of development factors as returned by `create_chl_coefs()`, computes the cumulative product from each development period to the tail.

Usage

```
create_product_coefs(chl_coefs, name = "Product")
```

Arguments

chl_coefs A one-row data frame whose first column is a label and whose other columns hold development factors. (As returned by `create_chl_coefs()`.)
name Character – label for the resulting row.

Value

A character vector of cumulative-product factors with `name` as the first element.

create_run_off_check *Run-off check across all diagonals*

Description

Compares each diagonal's lower-tail cumulative ultimate against the main diagonal's. Returns the run-off differences in chronological order.

Usage

```
create_run_off_check(cum_trg)
```

Arguments

cum_trg A cumulative numeric triangle.

Value

Numeric vector of run-off differences with length = nrow(cum_trg) - 1.

create_triangle *Build a development triangle from row-level claims data*

Description

Aggregates value over (row_num, col_num) (optionally after filtering by one or more cond_variable == cond_value pairs) and arranges the result into a square rows x rows matrix where the upper-right corner beyond the main anti-diagonal is NA.

Usage

```
create_triangle(
  data,
  row_num,
  col_num,
  value,
  cond_variable = NULL,
  cond_value = NULL,
  rows = NULL
)
```

Arguments

data	A data frame (or tibble) with at least the columns named by row_num, col_num and value.
row_num	Character - name of the column with row indices (origin periods, 1-based).
col_num	Character - name of the column with column indices (development periods, 0-based).
value	Character - name of the numeric column to be summed.
cond_variable	Optional character vector - names of columns to filter on.
cond_value	Optional vector of values, same length as cond_variable - equality filter applied pairwise.
rows	Optional integer - size of the output triangle. Defaults to the maximum value of data[[row_num]].

Value

A numeric rows x rows matrix.

Examples

```
df <- expand.grid(origin = 1:3, dev = 0:2)
df$paid <- c(100, 50, 20, 120, 40, NA, 130, NA, NA)
df <- df[!is.na(df$paid), ]
create_triangle(df, row_num = "origin", col_num = "dev", value = "paid")
```

diag_reader

Read values along a triangle diagonal

Description

Returns the values along the diag_num-th anti-diagonal of trg, starting from the bottom-left.

Usage

```
diag_reader(trg, diag_num = nrow(trg))
```

Arguments

trg	A square numeric matrix (a triangle).
diag_num	Integer - which anti-diagonal to read. Defaults to the main (last) one.

Value

A numeric vector with the values on that diagonal.

Examples

```
trg <- matrix(1:9, nrow = 3)
diag_reader(trg)
diag_reader(trg, diag_num = 2)
```

diag_sums

Cumulative diagonal sums of a triangle

Description

For each anti-diagonal $k = 1 \dots \text{nrow}(\text{trg})$ returns the sum of its values.

Usage

```
diag_sums(trg)
```

Arguments

trg A square numeric matrix (a triangle).

Value

Numeric vector of length $\text{nrow}(\text{trg})$.

Examples

```
diag_sums(matrix(c(10, 5, 2, 20, 7, NA, 15, NA, NA), nrow = 3, byrow = TRUE))
```

diag_writer

Append the latest diagonal of a new triangle to an old one

Description

Used in rolling re-runs: pads `old_trg` with one extra row and column of NA, then fills in the gaps from `new_trg` (which already contains the newly observed diagonal).

Usage

```
diag_writer(old_trg, new_trg)
```

Arguments

old_trg A $(n) \times (n)$ matrix - the old triangle.

new_trg A $(n+1) \times (n+1)$ matrix - the new triangle including the freshly observed diagonal.

Value

The merged $(n+1) \times (n+1)$ matrix.

Examples

```
old <- matrix(c(10, 5, 20, NA), nrow = 2, byrow = TRUE)
new <- matrix(c(10, 5, 3, 20, 6, NA, 25, NA, NA), nrow = 3, byrow = TRUE)
diag_writer(old, new)
```

get_colors

Pick n evenly spaced colors from a rainbow data frame

Description

Three flavours are exported. They differ only in the post-processing they apply after picking equally-spaced rows from `rgb_df`:

Usage

```
get_colors(rgb_df, n_colors)

get_colors_plus(rgb_df, n_colors)

get_colors_duo(rgb_df, n_colors)
```

Arguments

`rgb_df` A data frame with columns `r`, `g`, `b`, e.g. produced by `rgb_colors_for_plot()`.
`n_colors` Integer ≥ 2 – number of colors to return.

Details

- `get_colors()` – no post-processing.
- `get_colors_plus()` – every other color is darkened ($\times 0.75$) or lightened (towards 255 with weight 0.7), creating a strong duo pattern.
- `get_colors_duo()` – every other color is darkened ($\times 0.8$) or lightened (weight 0.6), milder than the `_plus` variant.

Value

A character vector of length `n_colors` with hex color codes ("`#RRGGBB`").

Examples

```
rgb_df <- rgb_colors_for_plot()
get_colors(rgb_df, 5)
get_colors_plus(rgb_df, 6)
get_colors_duo(rgb_df, 6)
```

load_excel_sheets *Load every sheet of an Excel .xlsx workbook into a named list*

Description

Wrapper around `readxl::excel_sheets()` and `readxl::read_xlsx()`. Returns a list whose names are the sheet names and whose elements are the parsed tibbles.

Usage

```
load_excel_sheets(path, range = NULL)
```

Arguments

`path` Path to the workbook.
`range` Optional cell range passed to `readxl::read_xlsx()` (e.g. "A1:D20").

Value

A named list of data frames – one entry per sheet.

Examples

```
## Not run:
sheets <- load_excel_sheets("path/to/file.xlsx")
sheets <- load_excel_sheets("path/to/file.xlsx", range = "A1:F100")

## End(Not run)
```

load_xlsb_sheets *Load every sheet of a binary Excel .xlsb workbook into a named list*

Description

Wrapper around the `readxlsb` package. Returns a list whose names are the sheet names and whose elements are the parsed data frames.

Usage

```
load_xlsb_sheets(path, range = NULL)
```

Arguments

`path` Path to the workbook.
`range` Optional cell range passed to `readxlsb::read_xlsb()`.

Value

A named list of data frames – one entry per sheet.

Note

readxlsb was archived from CRAN in 2024-09-25. To use this function you have to install it from GitHub manually: `remotes::install_github("velofrog/readxlsb")`.

Examples

```
## Not run:  
# Requires: remotes::install_github("velofrog/readxlsb")  
sheets <- load_xlsb_sheets("path/to/file.xlsb")  
  
## End(Not run)
```

numeric_format	<i>Format a number with a thin space as the thousands separator</i>
----------------	---

Description

Convenience wrapper around `base::format()` with sensible Czech-friendly defaults (thin space as group separator, `.` as decimal mark, no scientific notation).

Usage

```
numeric_format(number)
```

Arguments

number Numeric vector.

Value

A character vector of formatted numbers.

Examples

```
numeric_format(c(1234567, 89.5))
```

plot_colorBars	<i>Quick visual sanity check of a generated palette</i>
----------------	---

Description

Convenience wrapper that calls `rgb_colors_for_plot()` and `get_colors_duo()` and renders the result as a stacked bar chart.

Usage

```
plot_colorBars(n_colors)
```

Arguments

`n_colors` Integer ≥ 2 – number of palette stops to plot.

Value

Invisible NULL. Called for the side effect of plotting.

Examples

```
plot_colorBars(12)
```

rainbow_cat	<i>Print colored text to the console (rainbow cycle)</i>
-------------	--

Description

Prints `print_text` to the console wrapped in an ANSI color code chosen by `color_num`. The color index cycles modulo 8 so any positive integer is valid.

Usage

```
rainbow_cat(print_text, color_num)
```

Arguments

`print_text` Character – the text to print.
`color_num` Integer – color index (cycled modulo 8).

Value

Invisible NULL. Called for the side effect of printing.

Examples

```
rainbow_cat("Hello in green", 3)
```

rc_to_birth_day	<i>Convert a Czech birth number to a date of birth</i>
-----------------	--

Description

These three variants implement slightly different rules for resolving the century (the two-digit year stored in a Czech *rodne cislo*). They are kept as separate functions so existing code that depends on a particular rule keeps working.

Usage

```
rc_to_birth_day(rc_s_lomitkem)
```

```
rc_to_birth_day_2(rc_s_lomitkem)
```

```
rc_to_birth_day_3(rc_s_lomitkem)
```

Arguments

`rc_s_lomitkem` Character vector. Czech birth number, optionally with a / separating the suffix. The first six digits (YYMMDD) are what matters for the date.

Details

- `rc_to_birth_day()` - assumes the input has the trailing 4-digit suffix separated by a slash ("YYMMDD/XXXX", total length 11). Years < 54 go into the 21st century, otherwise the 20th.
- `rc_to_birth_day_2()` - does not check length and uses cut-off 25 - anything < 25 is 21st century, otherwise 20th.
- `rc_to_birth_day_3()` - like `rc_to_birth_day()` but with cut-off 25.

All three apply the women-offset (+50 added to the month for women, so months 51-62 -> 01-12).

Value

A Date vector of the same length as `rc_s_lomitkem`.

Examples

```
rc_to_birth_day("905615/1234") # man, 1990-06-15
rc_to_birth_day("9056151234") # man, 1990-06-15 (no slash, length 10)
rc_to_birth_day_2("055615/1234") # cut-off 25 -> 2005-06-15
rc_to_birth_day_3("055615/1234") # 11 chars + cut-off 25 -> 2005-06-15
```

rgb_colors_for_plot *Build a smooth RGB rainbow data frame*

Description

Produces a tibble of r, g, b channel values walking through the six canonical rainbow segments (red -> yellow -> green -> cyan -> blue -> purple -> red). The argument selects which segments to include and in which order.

Usage

```
rgb_colors_for_plot(colors_changes_vector = c(3:6, 1))
```

Arguments

colors_changes_vector
Integer vector with values in 1:6. Default c(3:6, 1) gives a green -> red sweep.

Value

A `tibble::tibble()` with columns r, g, b.

Examples

```
head(rgb_colors_for_plot())
```

triangle_namer *Rename a triangle's rows/columns for printing*

Description

Replaces the column names with 0..ncol-1 (development periods) and sets the row names to claim_period_names[1:nrow(trg)] (origin periods).

Usage

```
triangle_namer(trg, claim_period_names)
```

Arguments

trg A square numeric matrix.
claim_period_names
Character vector of origin-period labels (must be at least nrow(trg) long).

Value

The same matrix with dimnames set.

Examples

```
trg <- matrix(1:9, 3, 3)
triangle_namer(trg, c("2022", "2023", "2024"))
```

Index

`base::as.Date()`, [12](#), [13](#)
`base::format()`, [19](#)
`bind_with_source`, [3](#)

`create_annual_triangle`, [3](#)
`create_avg_coefs`, [4](#)
`create_chl_coef_triangle`, [4](#)
`create_chl_coef_triangle()`, [4](#)
`create_chl_coefs`, [5](#)
`create_chl_coefs()`, [6](#), [13](#)
`create_chl_coefs_weighted`, [6](#)
`create_chl_coefs_weighted()`, [6](#)
`create_chl_coefs_weighted_future`, [6](#)
`create_chl_trg_visualization`, [7](#)
`create_chl_trg_visualization()`, [7](#)
`create_chl_trg_visualization_pvzp`, [7](#)
`create_cumulative_triangle`, [8](#)
`create_cumulative_triangle()`, [9](#)
`create_decumulative_triangle`, [9](#)
`create_find_columns`, [9](#)
`create_multiple_columns_m`, [10](#)
`create_policy_exposure_columns_m`, [11](#)
`create_policy_exposure_columns_m()`, [12](#)
`create_policy_exposure_days_columns`,
[12](#)
`create_product_coefs`, [13](#)
`create_run_off_check`, [14](#)
`create_triangle`, [14](#)

`data.table::rbindlist()`, [3](#)
`data.table::setDT()`, [10](#)
`diag_reader`, [15](#)
`diag_sums`, [16](#)
`diag_writer`, [16](#)

`get_colors`, [17](#)
`get_colors_duo` (`get_colors`), [17](#)
`get_colors_duo()`, [20](#)
`get_colors_plus` (`get_colors`), [17](#)

`load_excel_sheets`, [18](#)

`load_xlsb_sheets`, [18](#)

`numeric_format`, [19](#)

`plot_color_bars`, [20](#)

`rainbow_cat`, [20](#)
`rc_to_birth_day`, [21](#)
`rc_to_birth_day_2` (`rc_to_birth_day`), [21](#)
`rc_to_birth_day_3` (`rc_to_birth_day`), [21](#)
`readxl::excel_sheets()`, [18](#)
`readxl::read_xlsx()`, [18](#)
`rgb_colors_for_plot`, [22](#)
`rgb_colors_for_plot()`, [17](#), [20](#)

`tibble::tibble()`, [22](#)
`triangle_namer`, [22](#)