

Package ‘R.Alpha.Home’

February 11, 2025

Type Package

Title Feel at Home using R, Thanks to Shortcuts Functions Making it Simple

Version 1.0.0

Description A collection of personal functions designed to simplify and streamline common R programming tasks. This package provides reusable tools and shortcuts for frequently used calculations and workflows.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Imports data.table, lubridate, rstudioapi, stringr, grDevices, openxlsx, ggplot2, dplyr, tibble, tidyr, R.utils, shinyWidgets, shiny, magrittr

NeedsCompilation no

Author Raphaël Flambard [aut, cre],
Adrien Cocuaud [ctb]

Maintainer Raphaël Flambard <r.alpha.act@outlook.fr>

Repository CRAN

Date/Publication 2025-02-11 09:20:02 UTC

Contents

cols_pad	2
compareVars	3
countSwitches	4
foldAllBr	5
importAll	6
left_join_checks	7
lum_0_100	9
quickSave	9
rdate	10
ret_lum	11

root	12
sepThsd	12
shiny_lum_0_100	13
timer	13

Index	15
--------------	-----------

cols_pad	<i>Add Variables to ease data usage in a Pivot Table</i>
----------	--

Description

Adds dummy columns to reach the number specified by the user. this is mostly useful to ensure straightforward and easy data updating when using pivot tables in Excel. It allows replacement of the previous data sheet by the new one, without having to take care about the number of columns, which will always be the same.

Usage

```
cols_pad(data, nCols = 100, colPrefix = "x_")
```

Arguments

data	The data frame to which dummy columns will be added.
nCols	the total number of columns required : default is 100
colPrefix	A string used as the prefix for the names of dummy columns.

Value

A data frame with the specified total number of columns.

Examples

```
table <- data.frame(a = 1:5, b = letters[1:5])
extraTable <- cols_pad(table, nCols = 6, colPrefix = "extra_")
print(extraTable)
```

`compareVars`*Compare Table Variables*

Description

Compares column names in two tables based on a given pattern. Provides information about which columns are present in which tables.

Usage

```
compareVars(x, y, pattern = "")
```

Arguments

<code>x</code>	A data frame representing the first table.
<code>y</code>	A data frame representing the second table.
<code>pattern</code>	A string pattern used to filter and compare only a subset of variables (column names).

Value

A list containing:

- `all`: All column names from both tables.
- `common`: Column names found in both tables.
- `onlyX`: Column names found only in the first table (`x`).
- `onlyY`: Column names found only in the second table (`y`).
- `exclusive`: Column names found in only one of the two tables.

Examples

```
# Example tables
table1 <- data.frame(exclusive_1 = 1:5, common_1 = 6:10, common_2 = 11:15)
table2 <- data.frame(common_1 = 16:20, common_2 = 21:25, exclusive_2 = 26:30)

# Compare all columns (no pattern given)
compare_all <- compareVars(table1, table2)
compare_all$common
compare_all$exclusive
compare_all$onlyX
compare_all$onlyY

# compare only columns following a specific pattern
compare_wPattern <- compareVars(table1, table2, pattern = "1")
compare_wPattern$all
compare_wPattern$common
```

countSwitches *create an incremented Counter, based on Start/Stop Markers*

Description

This function aims at identifying sections and sub-sections numbers, based on markers of section starts and ends.

Given a data frame, and the name of a column giving the start/stop markers, it will add columns giving infos about the successive section levels

Usage

```
countSwitches(
  data,
  colNm,
  sttMark,
  endMark,
  includeStt = TRUE,
  includeEnd = TRUE
)
```

Arguments

data	A data frame containing the column to process.
colNm	A string specifying the column name in 'data' to evaluate.
sttMark	A value indicating the start of a series.
endMark	A value indicating the end of a series.
includeStt	Logical. Should the start marker be included as part of the series? Default is 'TRUE'.
includeEnd	Logical. Should the end marker be included as part of the series? Default is 'TRUE'.

Value

A modified version of the input data frame with additional columns including:

- 'catLvl': The current series level calculated as the difference between the cumulative counts of start and end markers.
- 'lvl_1', 'lvl_2', 'lvl_3': Final series counts returned for each respective level.

Note

This function is currently mostly useful internally, to perform foldAllBr().

Examples

```
# example code
library(dplyr)
tribble(
  ~step
  , "start"
  , "content of section 1"
  , "start"
  , "subsection 1.1"
  , "end"
  , "end"
  , "out of any section"
  , "start"
  , "section 2"
  , "start"
  , "subsection 2.1"
  , "end"
  , "start"
  , "subsection 2.2"
  , "end"
  , "end"
) %>%
countSwitches(colNm = "step", "start", "end")
```

foldAllBr

Easily Fold Code Parts

Description

This function works with code split into parts identified by brackets. The format is as follows:

```
{
  ...
  code from part 1
  ...
} # part 1
{
  ...
} # part 2
```

It automatically identifies parts to fold/unfold easily.

Shortcuts required:

- "fold all brackets": shift + alt + S (Windows) / ctrl + shift + up (Mac)
- "expand fold": shift + alt + D (Windows) / ctrl + shift + down (Mac)

Usage

```
foldAllBr(time = FALSE, debug_getTbl = FALSE)
```

Arguments

time	Logical. If 'TRUE', the function will return ggplot object visualizing execution times for each step.
debug_getTbl	Logical. If 'TRUE', returns the 'docContent' table with tags for debugging purposes.

Value

A list containing:

- debug_info: A data frame with debugging information if debug_getTbl = TRUE.
- timer_plot: A ggplot object visualizing execution times if time = TRUE.

If both parameters are FALSE, the function returns a list with NULL values.

importAll

Function to Import and Concatenate Multiple data files

Description

Imports multiple files into a list, concatenates them into a single table, and adds an 'fName' variable. The files can be selected either by giving a file list (character vector), or by specifying a pattern.

Usage

```
importAll(
  path = ".",
  pattern = "",
  ignore.case = FALSE,
  importFunction = NULL,
  fill = FALSE,
  fileList = NULL
)
```

Arguments

path	Path to the directory, passed to 'list.files'.
pattern	Pattern to match file names, passed to 'list.files'.
ignore.case	Logical. If 'TRUE', ignores case when matching file names. Passed to 'list.files'. Default behavior is case-sensitive ('FALSE')
importFunction	A custom function for importing files. If not set, the function selects an import method based on the file extension.
fill	Logical. Passed to 'rbind' to allow filling missing columns.
fileList	A character vector of file names to import (used instead of 'pattern').

Value

A data frame containing the concatenated table with the fName column

Examples

```
# Directory containing test files
test_path <- tempdir()

# Create test files
write.csv( data.frame(a = 1:3, b = 4:6)      , file.path(test_path, "file1.csv"))
write.csv( data.frame(a = 7:9, b = 10:12)    , file.path(test_path, "file2.csv"))
write.csv( data.frame(a = 3:5, b = 8:10)     , file.path(test_path, "file3.csv"))
saveRDS(  data.frame(a = 1:5, b = 6:10)     , file.path(test_path, "file1.rds"))
saveRDS(  data.frame(a = 11:15, b = 16:20)  , file.path(test_path, "file2.rds"))

# Example 1 : Import all csv files
result <- importAll(path = test_path, pattern = "\\*.csv$")
print(result)

# Example 2: Import only selected files
file_list <- c("file1.csv", "file2.csv")
result <- importAll(path = test_path, fileList = file_list)
print(result)

# Example 3: Import all .rds files
result <- importAll(path = test_path, pattern = "\\*.rds$")
print(result)

# Example 4: Use a custom import function
custom_import <- function(file) {
  data <- read.csv(file, stringsAsFactors = FALSE)
  return(data)
}
result <- importAll(path = test_path, pattern = "\\*.csv$", importFunction = custom_import)
print(result)
```

left_join_checks

Left Join with Validation Checks

Description

a custom usage of left_join, with more detailed checks. Performs a left join and verifies that no unexpected duplicates or mismatches occur. In cas of unexpected results, gives details about what caused the problem.

Usage

```
left_join_checks(
  x,
  y,
  ...,
  req_xAllMatch = 1,
  req_preserved_x = 1,
  behavior = "error",
  showNotFound = FALSE,
  showProblems = TRUE,
  time = FALSE
)
```

Arguments

<code>x</code>	A <code>data.table</code> representing the left table.
<code>y</code>	A <code>data.table</code> representing the right table.
<code>...</code>	Additional arguments passed to <code>'dplyr::left_join'</code> .
<code>req_xAllMatch</code>	Logical. Ensure that all rows in <code>'x'</code> find a match in <code>'y'</code> . Default: <code>FALSE</code> .
<code>req_preserved_x</code>	Logical. Ensure that the number of rows in <code>'x'</code> remains unchanged after the join. Default: <code>TRUE</code> .
<code>behavior</code>	Character. Specifies behavior if validation fails. Options: <code>"warning"</code> or <code>"error"</code> . (default: <code>"warning"</code>)
<code>showNotFound</code>	Logical. Show rows from <code>'x'</code> that did not match with <code>'y'</code> . Default: <code>FALSE</code> .
<code>showProblems</code>	Logical. Display the problems encountered during the joining process, if any.
<code>time</code>	Logical. Internal argument used only for testing purposes, timing the function steps

Value

A `data.table` containing the joined table.

Examples

```
library(data.table)
library(dplyr)

# Example 1: Simple left join with all matches
table_left <- data.table(id = 1:3, value_left = c("A", "B", "C"))
table_right <- data.table(id = 1:3, value_right = c("X", "Y", "Z"))
result <- left_join_checks(table_left, table_right, by = "id", req_preserved_x = TRUE)
print(result) # Ensures all rows in table_left are preserved

# Example 2: Left join with missing matches
table_left <- data.table(id = 1:5, value_left = c("A", "B", "C", "D", "E"))
table_right <- data.table(id = c(1, 3, 5), value_right = c("X", "Y", "Z"))
```



```

result <- left_join_checks(
  table_left,
  table_right,
  by = "id",
  req_preserved_x = TRUE,
  showNotFound = TRUE,
  behavior = "warning"
)
print(result) # Rows from table_left with no matches in table_right are shown

```

lum_0_100	<i>Adjust the Brightness of the Graphics Window for comfortable viewing when using ggplot2</i>
-----------	--

Description

Modifies the brightness level of the active graphics window by adjusting its background color.

This is especially useful when using dark RStudio themes, where a 100 graphic window creates an uncomfortable contrast.

Usage

```
lum_0_100(lum = NULL)
```

Arguments

lum	Numeric. Brightness level, ranging from 0 (completely dark) to 100 (maximum brightness).
-----	--

Value

no return value : only apply the theme_set() function

quickSave	<i>Save File in a Directory storing saves, prefixing it with current date</i>
-----------	---

Description

Saves a file with current date in its name in a sub directory located in the same directory as the original file. Optionally, a note is added after the file name.

Usage

```
quickSave(
  saveDir,
  filePath = NULL,
  saveNote = NULL,
  overwrite = FALSE,
  verbose = FALSE
)
```

Arguments

saveDir	Choose the directory used to store saves. Suggested : 'old'
filePath	Optional, if you want to save another file than the current one : full path of the file you want to save.
saveNote	An optional custom note to append to the file name for the save, allowing to keep track of why this save has been done.
overwrite	Logical. Should an existing save with the same name be overwritten? Default is 'FALSE'.
verbose	logical. If turned to 'TRUE', the save path is displayed

Value

the output value of the function used to copy file

rdate

Generate Random Dates, with a similar usage as the r functions*

Description

Generates a vector of random dates within a specified range. This function tries to replicate the usage of the r* functions from stats package, such as runif(), rpois(), ...

Usage

```
rdate(
  x,
  min = paste0(format(Sys.Date(), "%Y"), "-01-01"),
  max = paste0(format(Sys.Date(), "%Y"), "-12-31"),
  sort = FALSE,
  include_hours = FALSE
)
```

Arguments

x	Integer. Length of the output vector (number of random dates to generate).
min	Date. Optional. The minimum date for the range. Defaults to the 1st of January of the current year.
max	Date. Optional. The maximum date for the range. Defaults to the 31st of December of the current year.
sort	Logical. Should the dates be sorted in ascending order? Default is 'FALSE'.
include_hours	Logical. Should the generated dates include time? Default is 'FALSE' (dates only). this will slow down the function

Value

A vector of random dates of length 'x'.

Examples

```
# Generate 5 random dates between two specific dates, sorted
rdate(5, min = as.Date("2020-01-01"), max = as.Date("2020-12-31"), sort = TRUE)

# Generate 7 random datetime values (with hours)
rdate(7, include_hours = TRUE)
```

ret_lum

Adjust the Brightness of a Hex Color

Description

Modifies the brightness of a color by multiplying its RGB components by a specified factor.
Mostly for internal usage inside lum_0_100 function.

Usage

```
ret_lum(hexCol, rgbFact)
```

Arguments

hexCol	Character. The color to adjust, specified in hexadecimal format (e.g., "#FF5733").
rgbFact	Numeric. The luminosity factor : - use a factor between 0 and 1 to decrease luminosity - use a factor >1 to increase it The final Brightness value will be maintained between 0 and 1.

Value

A modified hex color in hexadecimal format.

Examples

```
# Example 1: Lightening a color
ret_lum("#FF5733", 1.5) # Returns a lighter version of the input color

# Example 2: Darkening a color
ret_lum("#FF5733", 0.7) # Returns a darker version of the input color
```

root	<i>Get Root Directory of Current Source File</i>
------	--

Description

Returns the directory path where the current source code file is located.

It is especially useful when the same source code is used by multiple users, each using his own environment, with different file paths.

the aim is to avoid writing full paths in raw text inside source codes.

Usage

```
root()
```

Value

A character string representing the absolute path of the directory containing the current source file.

sepThsd	<i>Quick Number Formatting with Custom Defaults</i>
---------	---

Description

A wrapper for the 'format' function, designed to format numbers with custom defaults for thousands separator, number of significant digits, and scientific notation.

Usage

```
sepThsd(x, big.mark = " ", digits = 1, scientific = FALSE)
```

Arguments

x	Numeric. The input values to format.
big.mark	Character. The separator for thousands (e.g., " " for "1 000" or ",", for "1,000"). Default is " ".
digits	Integer. The number of significant digits to display. Default is '1'.
scientific	Logical. Should the numbers be displayed in scientific notation? Default is 'FALSE'.

Value

A character vector of formatted numbers.

Examples

```
# Format with a comma as a thousands separator and 3 significant digits
sepThsd(1234567.89, big.mark = ",", digits = 3)
# Use scientific notation
sepThsd(1234567.89, scientific = TRUE)
```

shiny_lum_0_100	<i>Set Shiny Background and Sidebar Colors to a Chosen Shade of Grey</i>
-----------------	--

Description

Adjust the background color of a Shiny app's main body and sidebar based on a specified luminosity level.

The purpose is the same as `lum_0_100()` function, avoiding problems with high contrast between with graphic windows and dark themes.

Usage

```
shiny_lum_0_100(lum)
```

Arguments

`lum` Numeric. Luminosity level, ranging from 0 (black) to 100 (white).

Value

The HTML tags for setting the background and sidebar colors.

timer	<i>allow organized tracking of R code execution time</i>
-------	--

Description

The 'timer' function allows you to append timeStamps to a data.table, and include additional meta-data provided as arguments. The last call calculates time differences between timeStamps.

Usage

```
timer(timer_table = data.table(), end = FALSE, ...)
```

Arguments

timer_table	A data.table containing the timer log to continue from. Defaults to an empty 'data.table().
end	A logical, indicating the end of the timer, defaulted to FALSE. 'timer()' calls must be placed at the beginning of each part : therefore, this 'closing' step is necessary to compute time for the last part. Time differences between timeStamps are calculated only when closing the timer.
...	Additional specifications. Use named arguments to provide documentation on the code parts you are timing : naming the current step, the version of the code you are trying, or any other useful specification

Value

A 'data.table' containing the original data, plus one new timeStamps, and optionally computed time differences :

- 'timeStamp': The current timeStamps ('POSIXct').
- 'timeStamp_num': timeStamps converted to numeric, useful for intermediary calculations.
- 'dt_num': The time difference in seconds between consecutive rows as a numeric value.
- 'dt_text': The formatted time difference in seconds with milliseconds as a character string.
- Additional columns for any information provided by the user via '...'. It allows documentation about the current step running, substeps, which version is being tested, ...

Examples

```
# compare code speed between using a loop, or the mean() function
library(data.table)
library(dplyr)
tmr <- data.table() # Initialize timer
vec <- rnorm(1e6)   # Example vector

tmr <- timer(tmr, method = "loop") # timeStamp : 1st step =====
total <- 0
for (i in seq_along(vec)) total <- total + vec[i]
mean_loop <- total / length(vec)

tmr <- timer(tmr, method = "mean()") # timeStamp : 1st step =====
mean_func <- mean(vec)

tmr <- timer(tmr, end = TRUE)       # timeStamp : close timer =====

t_step1 <- tmr[method == "loop"]$dt_num
t_step2 <- tmr[method == "mean()"]$dt_num
diff_pc <- (t_step2/t_step1 - 1) * 100
diff_txt <- format(diff_pc, nsmall = 0, digits = 1)

# view speed difference
print(tmr %>% select(-matches("_num$")))
paste0("speed difference : ", diff_txt, "%")
```

Index

cols_pad, 2
compareVars, 3
countSwitches, 4

foldAllBr, 5

importAll, 6

left_join_checks, 7
lum_0_100, 9

quickSave, 9

rdate, 10
ret_lum, 11
root, 12

sepThsd, 12
shiny_lum_0_100, 13

timer, 13