Package 'instatExtras'

December 3, 2024

Title A set of functions used in R-Instat

Version 0.2.0

Description This package contains a set of functions used in R-Instat. This includes functions written specifically for R-Instat, as well functions written elsewhere that are modified for use in R-Instat.

License LGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports digest,
dplyr,
forcats,
getPass,
ggplot2,
ggrepel,
gh,
grDevices,
gt,
htmlwidgets,
httr,
jsonlite,
magrittr,
methods,
plyr,
pingr,
rlang,
sjlabelled,
stringr,
tidyr,
tidyselect

Contents

cancor_coef *Cancor coef*

Description

: This function takes an object as input and returns the "xcoef" and "ycoef" elements from the object.

Usage

```
cancor_coef(object)
```

Arguments

object	An object containing "xcoef" and "ycoef" elements.

Value

: A subset of the object containing only the "xcoef" and "ycoef" elements.

Examples

```
data <- list(xcoef = c(0.5, 0.3, -0.2), ycoef = c(0.8, -0.4, 0.6))
cancor_coef(data)
# Returns:
# $xcoef
# [1] 0.5 0.3 -0.2
#
# $ycoef
# [1] 0.8 -0.4 0.6
```

cbind_unique *Bind Data Frames and Remove Duplicates*

Description

This function binds two data frames and removes duplicates from the first data frame based on the columns in the second data frame.

Usage

```
cbind_unique(x, y, cols)
```

Arguments

x	A data frame from which duplicates will be removed.
y	A data frame containing the columns to bind into x.
cols	A character vector of column names to use for binding.

Value

A data frame with the bound columns and duplicates removed.

Examples

```
## Not run:
  cbind_unique(df1, df2, c("col1", "col2"))

## End(Not run)
```

`check_github_repo` *Check GitHub Repository*

Description

Verifies the existence and status of a GitHub repository, including whether it is an R package, and checks if a locally installed package is up-to-date with the latest commit on GitHub.

Usage

```
check_github_repo(owner = NULL, repo = NULL, url = NULL)
```

Arguments

`owner`	A character string specifying the GitHub repository owner. Defaults to `NULL`.
`repo`	A character string specifying the repository name. Defaults to `NULL`.
`url`	A character string specifying the full GitHub URL of the repository. Defaults to `NULL`.

Value

An integer status code:

0 Package is installed and up-to-date.

1 Package is installed but not the latest version.

2 Unable to retrieve the latest commit from GitHub.

3 Package is installed but not from GitHub.

4 Repository exists and is an R package.

5 Repository exists but is not an R package.

6 Repository does not exist or an error occurred.

Examples

```
# Check a repository using owner and repo
check_github_repo(owner = "hadley", repo = "ggplot2")

# Check a repository using a URL
check_github_repo(url = "https://github.com/hadley/ggplot2")
```

check_graph *Check Graph*

Description

Records the plot if `graph_object` is NULL and returns the graph object of class "recordedplot". Applicable to base graphs only.

Usage

```
check_graph(graph_object)
```

Arguments

`graph_object`	The graph object to be checked.

Value

The recorded plot object or NULL if an error occurs.

Examples

```
## Not run:
  check_graph(my_graph)

## End(Not run)
```

compare_columns *compare columns Compare two columns and provide information about their values*

Description

compare columns Compare two columns and provide information about their values

Usage

```
compare_columns(
  x,
  y,
  use_unique = TRUE,
  sort_values = TRUE,
  firstnotsecond = TRUE,
  secondnotfirst = TRUE,
  display_intersection = FALSE,
  display_union = FALSE,
  display_values = TRUE
)
```

Arguments

x	The first column to compare.
y	The second column to compare.
use_unique	If TRUE, remove duplicate values from both columns before comparison. Default is TRUE.
sort_values	If TRUE, sort the values in both columns before comparison. Default is TRUE.
firstnotsecond	If TRUE, display the values that appear in the first column but not in the second column. Default is TRUE.
secondnotfirst	If TRUE, display the values that appear in the second column but not in the first column. Default is TRUE.
display_intersection	If TRUE, display the values that appear in both columns. Default is FALSE.
display_union	If TRUE, display the values that appear in either column. Default is FALSE.
display_values	If TRUE and the columns contain the same values, display the values. Default is TRUE.

Value

None.

Examples

```
x <- 1:10
y <- 5:14
compare_columns(x, y)
```

---

consecutive_sum *Consecutive Sum*

Description

This function calculates the consecutive sum of a given vector.

Usage

```
consecutive_sum(x, initial_value = NA)
```

Arguments

x	A numeric vector.
initial_value	A numeric value representing the initial sum value. Default is NA.

Value

A numeric vector with the consecutive sum values.

Examples

```
consecutive_sum(c(1, 2, 3, 4, 5))
# Output: 1 3 6 10 15

consecutive_sum(c(1, NA, 3, 0, 5), initial_value = 10)
# Output: 11 NA 14 14 19
```

---

convert_to_list *Convert Character to List of Numeric Vectors*

Description

Converts a character string to a list of numeric vectors.

Usage

```
convert_to_list(x)
```

Arguments

x	A character string to be converted.

Value

A list of numeric vectors.

Examples

```
## Not run:
  convert_to_list("c(1,2,3)")

## End(Not run)
```

---

create_av_packs *Create Available Packages Data Frame*

Description

This function retrieves the list of available packages from the specified repository and stores it in a data frame.

Usage

```
create_av_packs()
```

Value

A data frame containing the list of available packages.

Examples

```
create_av_packs()
```

`drop_unused_levels` *Drop unused levels*

Description

Drops unused levels from specified columns in a data frame.

Usage

```
drop_unused_levels(data, columns)
```

Arguments

`data`	A data frame.
`columns`	A character vector specifying the column names in `data` from which unused levels should be dropped.

Value

The modified data frame with unused levels dropped from the specified columns. If a column is not a factor, it remains unchanged.

Examples

```
# Create a data frame
df <- data.frame(A = factor(c("apple", "banana", "apple", "orange")),
                 B = factor(c("red", "blue", "green", "red")),
                 C = c(1, 2, 3, 4))

# Drop unused levels from column 'A'
drop_unused_levels(df, columns = "A")
# Output:
#       A    B C
# 1 apple  red 1
# 2 banana blue 2
# 3 apple green 3
# 4 orange  red 4

# Drop unused levels from multiple columns
drop_unused_levels(df, columns = c("A", "B"))
# Output:
#       A    B C
# 1 apple  red 1
# 2 banana blue 2
# 3 apple green 3
# 4 orange  red 4
```

duplicated_cases *Duplicated Cases*

Description

This function identifies duplicated cases in a column and assigns a unique identifier to each duplicate case. The function supports both numeric and non-numeric columns.

Usage

```
duplicated_cases(col_name, ignore = NULL, tolerance = 0.01)
```

Arguments

col_name	A vector representing the column for which duplicated cases need to be identified.
ignore	A vector of values to be ignored when identifying duplicated cases. These values will not be considered as duplicates.
tolerance	A numeric value representing the tolerance level for numeric columns. If the absolute difference between two consecutive numeric values is within this tolerance, they are considered duplicates.

Value

A vector of the same length as the input column, where each element represents a unique identifier for each case. Duplicated cases are assigned the same identifier.

Examples

```
#' # Example 1: Numeric column
col <- c(1, 2, 2.01, 3, 4, 4.005, 4.01, 4.02, 4.03)
duplicated_cases(col)
# Output: 1 1 2 1 1 2 3 4 5

# Example 2: Non-numeric column
col <- c("A", "A", "B", "C", "C", "D", "E")
duplicated_cases(col)
# Output: 1 1 1 1 2 1 1
```

duplicated_count_index *Calculate count or Index of duplicated Values*

Description

This function calculates either the count or index of duplicated values in a data frame.

Usage

```
duplicated_count_index(x, type = "count")
```

Arguments

x	A data frame or a vector that can be coerced into a data frame.
type	A character string specifying the type of calculation to perform. Possible values are "count" (default) or "index".

Value

If type = "count", it returns a vector with the count of duplicates for each row in the data frame x, excluding the row itself. If type = "index", it returns a vector with the index of duplicates for each row in the data frame x, where the index represents the number of previous duplicates (starting from 1) for each unique combination of values.

Examples

```
# Example 1: Count of duplicated rows
#data <- data.frame(a = c(1, 2, 3, 2, 1, 3, 4))
#duplicated_count_index(data, type = "count")

# Output:
# [1] 1 1 1 1 1 1 0

# Example 2: Index of duplicated rows
#data <- data.frame(a = c(1, 2, 3, 2, 1, 3, 4))
#duplicated_count_index(data, type = "index")

# Output:
# [1] 0 0 0 1 2 1 0
```

---

frac10 *Convert Decimal to Fraction*

Description

Converts a decimal value into a fraction with a specified denominator or common denominators (10, 20, or 100).

Usage

```
frac10(x)

frac20(x)

frac100(x)

frac_den(x, den)
```

Arguments

x	A numeric value representing the decimal to convert.
den	An integer specifying the denominator for the fraction. (Only for frac_den.)

Value

A character string representing the fraction.

Examples

```
# Convert decimals to fractions
frac10(0.75)  # "8/10"
frac20(0.25)  # "5/20"
frac100(0.123) # "12/100"
frac_den(0.333, 3) # "1/3"
```

---

getExample *Get Example*

Description

Retrieves and displays the example code for a specified topic from a package.

Usage

```
getExample(
  topic,
  package = NULL,
  lib.loc = NULL,
  character.only = TRUE,
  give.lines = FALSE,
  local = FALSE,
  echo = TRUE,
  verbose = getOption("verbose"),
  setRNG = FALSE,
  ask = getOption("example.ask"),
  prompt.prefix = abbreviate(topic, 6),
  run.dontrun = FALSE,
  run.donttest = interactive()
)
```

Arguments

topic	The topic for which to retrieve the example.
package	The name of the package containing the topic. Defaults to NULL.
lib.loc	The library location. Defaults to NULL.
character.only	Logical indicating whether the topic is specified as a character string. Defaults to TRUE.

give.lines	Logical indicating whether to return the example code as lines of text. Defaults to FALSE.
local	Logical indicating whether to evaluate the example locally. Defaults to FALSE.
echo	Logical indicating whether to echo the example code. Defaults to TRUE.
verbose	Logical indicating whether to print verbose output. Defaults to the value of the "verbose" option.
setRNG	Logical indicating whether to set the random number generator. Defaults to FALSE.
ask	Logical indicating whether to ask before evaluating the example. Defaults to the value of the "example.ask" option.
prompt.prefix	A prefix for the prompt. Defaults to an abbreviated version of the topic.
run.dontrun	Logical indicating whether to run examples marked with \dontrun. Defaults to FALSE.
run.donttest	Logical indicating whether to run examples marked with \donttest. Defaults to the value of interactive().

Value

The example code as a character string if give.lines is TRUE, otherwise prints the example code.

Examples

```
## Not run:
  getExample("filter", "dplyr")

## End(Not run)
```

---

getRowHeadersWithText *Get Row Headers with Text*

Description

Retrieves the row headers of a data frame where the specified column contains the search text.

Usage

```
getRowHeadersWithText(
  data,
  column,
  searchText,
  ignore_case,
  use_regex,
  match_entire_cell
)
```

Arguments

data	The input data frame.
column	The name of the column to search.
searchText	The text to search for.
ignore_case	Logical indicating whether to ignore case. Defaults to TRUE.
use_regex	Logical indicating whether to use regular expressions. Defaults to FALSE.
match_entire_cell	Logical indicating whether to match the entire cell.

Value

A character vector of row headers where the search text is found.

Examples

```
## Not run:
  getRowHeadersWithText(my_data, "column1", "search text", TRUE, FALSE)

## End(Not run)
```

---

get_column_attributes *Get column attributes*

Description

This function retrieves the attributes of a given object, excluding specified attributes to be dropped.

Usage

```
get_column_attributes(x, drop = c("class", "levels"))
```

Arguments

x	The object for which attributes need to be retrieved.
drop	A character vector specifying the names of attributes to be dropped from the result. Default is c("class", "levels").

Value

A list of attributes of the object, excluding the specified attributes to be dropped.

Examples

```
# Example 1: Get attributes of a vector
vec <- c(1, 2, 3)
get_column_attributes(vec)

# Output:
# $dim
# NULL
```

get_data_book_output_object_names

Get Data Book Output Object Names

Description

Retrieves the names of output objects in a data book list.

Usage

```
get_data_book_output_object_names(
  output_object_list,
  object_type_label = NULL,
  excluded_items = c(),
  as_list = FALSE,
  list_label = NULL
)
```

Arguments

output_object_list	A list of output objects.
object_type_label	An optional label to filter the objects by type. Defaults to NULL.
excluded_items	A character vector of items to exclude from the result. Defaults to an empty vector.
as_list	Logical indicating whether to return the result as a list. Defaults to FALSE.
list_label	An optional label for the list if as_list is TRUE. Defaults to NULL.

Value

A character vector or list of object names.

Examples

```
## Not run:
  get_data_book_output_object_names(my_objects)

## End(Not run)
```

get_data_book_scalar_names

Get Scalar Names from Data Book

Description

Extracts scalar names from a given list, with the option to exclude specific items, return the names as a list, and provide a label for the list.

Usage

```
get_data_book_scalar_names(
  scalar_list,
  excluded_items = c(),
  as_list = FALSE,
  list_label = NULL
)
```

Arguments

scalar_list	A named list from which to extract scalar names.
excluded_items	A character vector of items to exclude from the output. Defaults to an empty vector.
as_list	A logical value indicating whether to return the result as a list. Defaults to FALSE.
list_label	A character string specifying the label for the list, if as_list = TRUE.

Value

A character vector of scalar names, or a named list if as_list = TRUE.

Examples

```
# Extract names excluding specific items
get_data_book_scalar_names(list(a = 1, b = 2, c = 3), excluded_items = c("b"))

# Return the names as a list with a label
get_data_book_scalar_names(list(a = 1, b = 2), as_list = TRUE, list_label = "Scalars")
```

---

get_default_significant_figures

Get the default number of significant figures

Description

This function gets the default number of significant figures when given a numeric vector.

Usage

```
get_default_significant_figures(data)
```

Arguments

data	numeric(1) A numerical vector

Value

If the data is numeric, "3", otherwise NA.

Examples

```
x <- 1:8
get_default_significant_figures(x)
```

`get_installed_packages_with_data` *Get Installed Packages with Data*

Description

This function retrieves a list of installed packages in R, optionally including only those packages that contain data.

Usage

```
get_installed_packages_with_data(with_data = TRUE)
```

Arguments

`with_data`	Logical value indicating whether to include only packages that contain data. Default is `TRUE`.

Value

A character vector containing the names of the installed packages. If `with_data` is `TRUE`, only packages with data will be included.

Examples

```
# Get all installed packages
get_installed_packages_with_data()

# Get installed packages with data
get_installed_packages_with_data(with_data = TRUE)

# Get all installed packages (including those without data)
get_installed_packages_with_data(with_data = FALSE)
```

`get_odk_form_names` *Get ODK Form Names*

Description

This function takes a username and platform as input and retrieves the names of ODK forms from the specified platform. The supported platforms are "kobo" and "ona". The function authenticates the user with the platform if the username and password are provided. Otherwise, it retrieves the form names without authentication. The function returns a character vector containing the form names.

Usage

```
get_odk_form_names(username, platform)
```

Arguments

username	The username for authentication with the ODK platform. Optional if authentication is not required.
platform	The platform where the ODK forms are hosted. Supported values are "kobo" and "ona".

Value

A character vector containing the names of ODK forms retrieved from the specified platform.

Examples

```
# Get ODK form names from Kobo platform
# get_odk_form_names(username = "myusername", platform = "kobo")

# Get ODK form names from Ona platform
# get_odk_form_names(username = "myusername", platform = "ona")

# Get ODK form names without authentication
# get_odk_form_names(platform = "kobo")
```

---

get_vignette *Get Vignette*

Description

Retrieves the vignette information for a specified package.

Usage

```
get_vignette(package = NULL, lib.loc = NULL, all = TRUE)
```

Arguments

package	The name of the package. Defaults to NULL.
lib.loc	The library location. Defaults to NULL.
all	Logical indicating whether to list all available vignettes. Defaults to TRUE.

Value

A character vector with the vignette information.

Examples

```
## Not run:
  get_vignette("dplyr")

## End(Not run)
```

hashed_id *Hashed id*

Description

This function generates hashed identifiers by applying a specified algorithm to the input values, optionally incorporating a salt value.

Usage

```
hashed_id(x, salt, algo = "crc32")
```

Arguments

x	A vector or list of values to be hashed.
salt	A character string representing a salt value to be appended to each element in x. Default is NULL, indicating no salt.
algo	A character string specifying the hashing algorithm to be used. Default is "crc32".

Value

A character vector of hashed identifiers corresponding to the input values.

Examples

```
# Example 1: Generate hashed identifiers without salt
x <- c("apple", "banana", "cherry")
hashed_id(x)

# Output:
# [1] "AD5A2589" "77D3B209" "2C6EDC7A"

# Example 2: Generate hashed identifiers with salt
x <- c("apple", "banana", "cherry")
salt <- "salty"
hashed_id(x, salt)

# Output:
# [1] "1A1E6CE0E9A969A9" "1B69B1B8C1D6EF99" "1C10A25AA55025CD"
```

import_from_ODK *Import from ODK*

Description

This function imports data from ODK platforms, such as KoboToolbox or Ona, based on the specified username, form name, and platform.

Usage

```
import_from_ODK(username, form_name, platform)
```

Arguments

username	Your username (character) on the ODK platform.
form_name	The name of the form (character) you want to import
platform	The ODK platform (character)you are using. Valid options are "kobo" or "ona".

Value

The imported form data as a structured object.

---

in_top_n *Find values in the top N*

Description

This function takes a vector of values and checks if each value is among the top N values based on the specified criteria. The criteria can include weighting the values and applying a custom aggregation function to calculate a weighted score. The function returns a logical vector indicating whether each value is in the top N.

Usage

```
in_top_n(x, n = 10, wt, fun = sum)
```

Arguments

x	The vector of values to be checked.
n	The number of top values to consider. Default is 10.
wt	A vector of weights corresponding to the values. If provided, the values will be weighted before determining the top N.
fun	The aggregation function to be applied to calculate a weighted score. Default is sum.

Value

A logical vector indicating whether each value is in the top N.

Examples

```
# Check if values are in the top 5
# in_top_n(x = c(10, 5, 7, 12, 3), n = 5)

# Check if weighted values are in the top 3 based on the sum
# in_top_n(x = c(10, 5, 7, 12, 3), n = 3, wt = c(2, 1, 3, 4, 2), fun = sum)
```

is.binary *Checking Binary variable*

Description

This function checks if the input is a binary variable. It determines whether the input is of logical, numeric, or factor type and checks if it meets the criteria for a binary variable.

Usage

```
is.binary(x)
```

Arguments

x	The input variable to be checked for binary nature.

Value

The function returns a logical value indicating whether the input is binary or not. It returns TRUE if the input is binary, and FALSE otherwise.

Examples

```
is.binary(TRUE)           # TRUE
is.binary(c(0, 1, 1, 0))  # TRUE
is.binary(factor(c("Yes", "No", "Yes", "Yes")))  # TRUE
is.binary(c(1, 2, 3, 4))  # FALSE
```

is.containPartialValueLabel *Check for Partial Value Labels*

Description

Checks if the variable contains partial value labels (some values labeled, others not).

Usage

```
is.containPartialValueLabel(x)
```

Arguments

x	A variable to check for partial value labels.

Value

A logical value. Returns TRUE if the variable contains partial value labels, otherwise FALSE.

Examples

```
# Example with partially labeled variable
#is.containPartialValueLabel(x)
```

`is.containValueLabel` *Checking if an object contains a value label attribute*

Description

This function checks if an object contains a value label attribute. It determines whether the input object has an attribute with the name "labels_label".

Usage

```
is.containValueLabel(x)
```

Arguments

`x`	The object to be checked for the presence of a value label attribute.

Value

The function returns a logical value indicating whether the input object contains a value label attribute. It returns TRUE if the attribute is present, and FALSE otherwise.

Examples

```
#df <- data.frame(x = 1:10, y = 11:20)
#attributes(df)$labels_label <- "Value Labels"
#is.containValueLabel(df)  # TRUE

#vec <- c(1, 2, 3, 4, 5)
#is.containValueLabel(vec)  # FALSE
```

`is.containVariableLabel`
Checking if an object has a variable label attached.

Description

This function checks if an object has a variable label attached to it. It determines whether the input object `x` has a non-empty variable label using the `sjlabelled::get_label()` function.

Usage

```
is.containVariableLabel(x)
```

Arguments

x	The object to be checked for the presence of a variable label.

Value

The function returns a logical value indicating whether the input object contains a variable label. It returns TRUE if a non-empty variable label is found, and FALSE otherwise.

Examples

```
df <- data.frame(x = 1:10, y = 11:20)
sjlabelled::set_label(df$x, "Age")
is.containVariableLabel(df$x)

vec <- c(1, 2, 3, 4, 5)
is.containVariableLabel(vec)
```

---

is.emptyvariable *Check if Variable is Empty*

Description

This function checks if a variable is empty, i.e., if it contains only empty values.

Usage

```
is.emptyvariable(x)
```

Arguments

x	The variable to be checked.

Value

A logical value indicating whether the variable is empty or not.

Examples

```
is.emptyvariable(c("", "abc", ""))
# [1] FALSE

is.emptyvariable(c("", ""))
# [1] TRUE
```

is.levelscount *Check Number of Levels in a Factor*

Description

This function checks if a factor variable has a specific number of levels.

Usage

```
is.levelscount(x, n)
```

Arguments

x	The factor variable to be checked.
n	The expected number of levels.

Value

A logical value indicating whether the factor variable has the specified number of levels.

Examples

```
#' # Example 1: Validating a factor variable with the correct number of levels
#factor_var <- factor(c("A", "B", "C", "D"))
#is.levelscount(factor_var, 4)
# Expected output: TRUE

# Example 2: Validating a factor variable with an incorrect number of levels
#factor_var <- factor(c("A", "B", "C", "D"))
#is.levelscount(factor_var, 3)
# Expected output: FALSE
```

is.logical.like *Logical like*

Description

Check if an object is logical-like.

Usage

```
is.logical.like(x)
```

Arguments

x	An object to be checked.

Value

TRUE if the object is logical-like, FALSE otherwise.

Examples

```
is.logical.like(TRUE)
# Output: TRUE

is.logical.like(c(TRUE, FALSE))
# Output: TRUE

is.logical.like(1)
# Output: TRUE

is.logical.like(c(0, 1))
# Output: TRUE

is.logical.like("TRUE")
# Output: FALSE

is.logical.like(NULL)
# Output: FALSE
```

---

is.NAvariable *Is NA variable*

Description

Check if a variable consists entirely of NA values.

Usage

```
is.NAvariable(x)
```

Arguments

x	An object to be checked.

Value

TRUE if the variable consists entirely of NA values, FALSE otherwise.

Examples

```
is.NAvariable(c(NA, NA, NA))
# Output: TRUE

is.NAvariable(c(1, 2, 3))
# Output: FALSE

is.NAvariable(c(TRUE, FALSE, NA))
# Output: FALSE

is.NAvariable(NULL)
# Output: FALSE
```

make_factor *Make factor*

Description

Creates a factor variable from the input data, with optional specification of ordering.

Usage

```
make_factor(x, ordered = is.ordered(x))
```

Arguments

x	The input data to be converted into a factor variable.
ordered	Logical value indicating whether the resulting factor should be ordered or not.

Value

A factor variable generated from the input data. If the input data is already a factor, and the ordered parameter is consistent with the existing ordering, the input data is returned as is. Otherwise, the input data is converted into a factor variable with the specified ordering.

Examples

```
# Create a factor from a numeric vector
#make_factor(c(1, 2,3,3,2,2,1,1,1, 3, 2), ordered = TRUE)
# Output: 1 2 3 2
# Levels: 1 < 2 < 3

# Create a factor from a logical vector
#make_factor(c(TRUE, FALSE,FALSE,TRUE,TRUE, TRUE), ordered = FALSE)
# Output: TRUE FALSE TRUE
# Levels: FALSE TRUE

# Create a factor from a character vector
#make_factor(c("apple", "banana", "apple", "orange"), ordered = FALSE)
# Output: apple banana apple orange
# Levels: apple banana orange

# Create a factor from an unsupported data type
#make_factor(Sys.time(), ordered = FALSE)
# Output: Error: The input data type is not supported for factor creation.
```

monitor_memory *Monitor Memory Usage*

Description

Monitors and returns the current memory usage in megabytes (MB).

Usage

```
monitor_memory()
```

Value

A numeric value representing the memory usage in MB.

Examples

```
# Check memory usage
monitor_memory()
```

---

next_default_item *Generate the next default item name*

Description

This function generates a new item name based on the provided prefix. It is useful when creating default item names or ensuring uniqueness in a set of item names.

Usage

```
next_default_item(
  prefix,
  existing_names = c(),
  include_index = FALSE,
  start_index = 1
)
```

Arguments

prefix	A character string representing the prefix for the item name.
existing_names	A vector of existing item names. Defaults to an empty vector.
include_index	A logical value indicating whether to include an index number in the generated item name. Defaults to FALSE.
start_index	An integer indicating the starting index for generating the item name. Defaults to 1.

Value

A character string representing the generated item name.

Examples

```
next_default_item("item", c("item1", "item2", "item3"))

next_default_item("item", c("item1", "item2", "item3"), include_index = TRUE, start_index = 5)
```

package_check *Package Check*

Description

This function checks the status of a specified package in the current R environment. It verifies whether the package is installed and, if so, compares the installed version with the latest version available online from CRAN.

Usage

```
package_check(package)
```

Arguments

package	A character string specifying the name of the package to be checked.

Value

A named list with the following elements:

status_code An integer indicating the package status:

- `0` - Package name not found in CRAN (incorrect spelling or not a CRAN package).
- `1` - Package is installed and up-to-date information is available.
- `2` - Package is a CRAN package but not installed.
- `3` - Package is installed but not found in CRAN (non-CRAN package).
- `4` - Package is neither installed nor found in CRAN.
- `5` - No internet connection available to check CRAN versions.

version_comparison An integer comparing the installed and CRAN versions:

- `-1` - Installed version is older than the CRAN version.
- `0` - Installed version matches the CRAN version.
- `1` - Installed version is newer than the CRAN version.
- `NA` - Version comparison is not applicable (e.g., for non-CRAN packages).

installed_version The installed version of the package (if available), as a character string.

cran_version The latest version available on CRAN (if available), as a character string.

Examples

```
# Check package "dplyr"
package_check("dplyr")

# Check package "ggplot2"
package_check("ggplot2")
```

process_html_object *Process Individual HTML Object*

Description

Displays the HTML object in the R viewer. If the viewer is not available, saves the object as a file in the temporary folder.

Usage

```
process_html_object(html_object)
```

Arguments

html_object	The HTML object to be processed.

Value

The HTML object or the file name if saved as an HTML file.

Examples

```
## Not run:
  process_html_object(my_html_object)

## End(Not run)
```

read_corpora *Read and Process Corpora Data*

Description

This function reads and processes data from the rcorpora package. It handles various data types including data frames, vectors, matrices, and lists.

Usage

```
read_corpora(data)
```

Arguments

data	A data frame, list, or other object containing the data to be processed.

Value

A data frame with the processed data, including metadata and descriptions if available.

Examples

```
## Not run:
  read_corpora(my_data)

## End(Not run)
```

record_graph *Record a Graph*

Description

This function captures a graph generated by the 'x' object and returns it as a recorded plot.

Usage

```
record_graph(x)
```

Arguments

x	An object representing the graph to be recorded.

Value

A recorded plot of the graph generated by 'x'.

Examples

```
# Example 1: Record a scatter plot
data <- data.frame(x = rnorm(100), y = rnorm(100))
plot(data$x, data$y)
recorded_plot <- record_graph(1)

# Example 2: Record a bar plot
bar_data <- c(A = 10, B = 20, C = 15)
barplot(bar_data)
recorded_plot <- record_graph(2)
```

slope *Slope*

Description

Calculate the slope of a linear regression model.

Usage

```
slope(y, x)
```

Arguments

y	A numeric vector of response variable values.
x	A numeric vector of predictor variable values.

Value

The slope of the linear regression model.

Examples

```
y <- c(1, 2, 3, 4, 5)
x <- c(2, 4, 6, 8, 10)
slope(y, x)
```

`slopegraph` *Plot a Slopegraph a la Tufte*

Description

This function and documentation are taken from the "newggslopegraph" function in the CGPfunctions R package. There are slight modifications for use in R-Instat. This function creates a "slopegraph" as conceptualised by Edward Tufte. Slopegraphs are minimalist and efficient presentations of your data that can simultaneously convey the relative rankings, the actual numeric values, and the changes and directionality of the data over time. Takes a dataframe as input, with three named columns being used to draw the plot. Makes the required adjustments to the ggplot2 parameters and returns the plot.

Usage

```
slopegraph(
  data,
  x,
  y,
  colour,
  data_label = NULL,
  y_text_size = 3,
  line_thickness = 1,
  line_colour = "ByGroup",
  data_text_size = 2.5,
  data_text_colour = "black",
  data_label_padding = 0.05,
  data_label_line_size = 0,
  data_label_fill_colour = "white",
  reverse_x_axis = FALSE,
  remove_missing = TRUE
)
```

Arguments

`data`	a dataframe or an object that can be coerced to a dataframe. Basic error checking is performed, to include ensuring that the named columns exist in the dataframe.
`x`	a column inside the dataframe that will be plotted on the x axis. Traditionally this is some measure of time. The function accepts a column of class ordered, factor or character. NOTE if your variable is currently a "date" class you must convert before using the function with `as.character(variablename)`.
`y`	a column inside the dataframe that will be plotted on the y axis. Traditionally this is some measure such as a percentage. Currently the function accepts a column of type integer or numeric. The slopegraph will be most effective when the y variables are not too disparate.

`colour` a column inside the dataframe that will be used to group and distinguish different y variables.

`data_label` an optional column inside the dataframe that will be used as the label for the data points plotted. Can be complex strings and have `NA` values but must be of class `chr`. By default `y` is converted to `chr` and used.

`y_text_size` Optionally the font size for the Y axis labels to be displayed. y_text_size = 3 is the default must be a numeric.

`line_thickness` Optionally the thickness of the plotted lines that connect the data points. LineThickness = 1 is the default must be a numeric.

`line_colour` Optionally the color of the plotted lines. By default it will use the ggplot2 color palette for coloring by `colour`. The user may override with **one** valid color of their choice e.g. "black" (see colors() for choices) **OR** they may provide a vector of colors such as c("gray", "red", "green", "gray", "blue") **OR** a named vector like c("Green" = "gray", "Liberal" = "red", "NDP" = "green", "Others" = "gray", "PC" = "blue"). Any input must be character, and the length of a vector **should** equal the number of levels in `colour`. If the user does not provide enough colors they will be recycled.

`data_text_size` Optionally the font size of the plotted data points. DataTextSize = 2.5 is the default must be a numeric.

`data_text_colour` Optionally the font color of the plotted data points. `"black"` is the default can be either `colors()` or hex value e.g. "#FF00FF".

`data_label_padding` Optionally the amount of space between the plotted data point numbers and the label "box". By default very small = 0.05 to avoid overlap. Must be a numeric. Too large a value will risk "hiding" datapoints.

`data_label_line_size` Optionally how wide a line to plot around the data label box. By default = 0 to have no visible border line around the label. Must be a numeric.

`data_label_fill_colour` Optionally the fill color or background of the plotted data points. `"white"` is the default can be any of the `colors()` or hex value e.g. "#FF00FF".

`reverse_x_axis` logical, set this value to `TRUE` if you want to reverse the **factor levels** on the x scale.

`remove_missing` logical, by default set to `TRUE` so that if any `y` is missing **all rows** for that `colour` are removed. If set to `FALSE` then the function will try to remove and graph what data it does have. **N.B.** missing values for `x` and `colour` are never permitted and will generate a fatal error with a warning.

Value

a plot of type ggplot to the default plot device

Author(s)

Chuck Powell

References

Based on: Edward Tufte, Beautiful Evidence (2006), pages 174-176. This function and documentation are taken from the `"newggslopegraph"` function in the `CGPfunctions` R package. Full credit on this function goes to the authors of the `CGPfunctions` R package.

Examples

```
data <- data.frame(
          Year = rep(c("2020", "2021"), each = 3),
          Value = c(10, 20, 15, 12, 25, 18),
          Group = rep(c("A", "B", "C"), times = 2)
          )
# Use the slopegraph function
slopegraph(data, x = Year, y = Value, colour = Group)
```

---

slopegraph_theme *Slope graph theme*

Description

A function that generates a theme for slopegraph plots.

Usage

```
slopegraph_theme(x_text_size = 12)
```

Arguments

x_text_size	The font size for the x-axis text (default: 12).

Value

A list of theme elements for slopegraph plots.

Examples

```
# Example 1: Generate a slopegraph theme with default parameters
#theme <- slopegraph_theme()

# Example 2: Generate a slopegraph theme with custom x-axis text size
#theme <- slopegraph_theme(x_text_size = 14)
```

---

split_items_in_groups *Split items into groups*

Description

This function takes a vector of items and splits them into a specified number of groups. The number of items must be divisible by the number of groups, otherwise an error is thrown.

Usage

```
split_items_in_groups(items, num)
```

Arguments

items	A vector of items to be split into groups.
num	The number of groups to split the items into.

Value

A list containing the groups of items. Each element of the list represents a group and contains a subset of the original items.

Examples

```
 items <- c("A", "B", "C", "D", "E", "F", "G", "H")
num_groups <- 2
split_items_in_groups(items, num_groups)

# Output:
# [[1]]
# [1] "A" "B" "C" "D"
#
# [[2]]
# [1] "E" "F" "G" "H"
```

---

summary_sample *Calculate summary sample*

Description

A function that summarizes a sample from a vector.

Usage

```
summary_sample(x, size, replace = FALSE)
```

Arguments

x	A vector from which to draw a sample.
size	The size of the sample to be drawn.
replace	Logical indicating whether sampling should be done with replacement (default: FALSE).

Value

A summary of the sample drawn from the input vector.

Examples

```
# Example 1: Draw a sample from a vector
data <- c(1, 2, 3, 4, 5)
sample <- summary_sample(x = data, size = )

# Example 2: Draw a sample from a vector with replacement
data <- c("A", "B", "C", "D", "E")
sample <- summary_sample(x = data, size = 4, replace = TRUE)
```

time_operation *Time an Operation*

Description

Measures and prints the time taken to execute an expression.

Usage

```
time_operation(expr)
```

Arguments

expr	An R expression to evaluate and time.

Value

Prints the time taken for the operation.

Examples

```
# Time a simple operation
time_operation({ Sys.sleep(1); mean(1:100) })
```

view_graph_object *View Graph Object*

Description

Displays the graph object in the R viewer. If the viewer is not available, saves the object as a file in the temporary folder.

Usage

```
view_graph_object(graph_object)
```

Arguments

graph_object	The graph object to be displayed.

Value

The graph object or the file name if saved as an image.

Examples

```
## Not run:
  view_graph_object(my_graph)

## End(Not run)
```

view_html_object *View HTML Object*

Description

Displays the HTML object in the R viewer. If the viewer is not available, saves the object as a file in the temporary folder.

Usage

```
view_html_object(html_object)
```

Arguments

html_object	The HTML object to be displayed.

Value

The HTML object or the file name if saved as an HTML file.

Examples

```
## Not run:
  view_html_object(my_html)

## End(Not run)
```

view_object *View Object*

Description

Displays the object stored in a data book object.

Usage

```
view_object(data_book_object)
```

Arguments

data_book_object	A data book object containing the object to be displayed and its format.

Value

The file name if the object is saved to a file, otherwise NULL.

Examples

```
## Not run:
  view_object(my_data_book)

## End(Not run)
```

view_object_data *View Object Data*

Description

Displays the given object in a specified format. If no format is provided, the object is printed.

Usage

```
view_object_data(object, object_format = NULL)
```

Arguments

object	The object to be displayed.
object_format	The format in which to display the object ("image", "text", "html"). Defaults to NULL.

Value

A file name if the object is saved to a file, otherwise NULL.

Examples

```
## Not run:
  view_object_data(my_object, "text")

## End(Not run)
```

view_text_object *View Text Object*

Description

Displays the text object in the R viewer. If the viewer is not available, saves the object as a file in the temporary folder.

Usage

```
view_text_object(text_object)
```

Arguments

text_object	The text object to be displayed.

Value

The text object or the file name if saved as a text file.

Examples

```
## Not run:
  view_text_object(my_text)

## End(Not run)
```

Index