Package 'usefun'

October 11, 2019

Type Package

Title A Collection of Useful Functions by John

Version 0.4.1

Description A set of general functions that I have used in various
projects and in other R packages. They support some miscellaneous operations
on data frames, matrices and vectors: adding a row on a ternary (3-value)
data.frame based on positive and negative vector-indicators, rearranging a
list of data.frames by rownames, pruning rows or columns of a data.frame
that contain only one specific value given by the user, checking for matrix equality,
pruning and reordering a vector according to the common elements between its
names and elements of another given vector, finding the non-common elements
between two vectors (outer-section),
normalization of a vector, matrix or data.frame's numeric values in a specified range,
pretty printing of vector names and values in an R notebook (common names and
values between two vectors also supported), retrieving the parent directory of
any string path, checking whether a numeric value is inside a given interval,
trim the decimal points of a given numeric value, quick saving of data to a file,
making a multiple densities plot and a color bar plot and executing a plot
string expression while generating the result to the specified file format.

License MIT + file LICENSE

URL https://github.com/bblodfon/usefun

BugReports https://github.com/bblodfon/usefun/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports grDevices, graphics, stats, utils

Suggests testthat, covr

NeedsCompilation no

Author John Zobolas [aut, cph, cre] (<https://orcid.org/0000-0002-3609-8674>)

Maintainer John Zobolas <bblodfon@gmail.com>

Repository CRAN

Date/Publication 2019-10-11 11:40:02 UTC

R topics documented:

`add_row_to_ternary_df` *Add a row to a 3-valued (ternary)* `data.frame`

Description

Use this function on a `data.frame` object (with values only in the 3-element set {-1,0,1} ideally - specifying either a positive, negative or none/absent condition/state/result about something) and add an extra **first or last row vector** with zero values, where *1* and *-1* will be filled when the column names of the given `data.frame` match the values in the *values.pos* or *values.neg* vector parameters respectively.

Usage

```
add_row_to_ternary_df(df, values.pos, values.neg, pos = "first",
  row.name = NULL)
```

Arguments

df	a data.frame object with values only in the the 3-element set {-1,0,1}. The column names should be node names (gene, protein names, etc.).
values.pos	a character vector whose elements are indicators of a positive state/condition and will be assigned a value of *1*. These elements **must be a subset of the column names** of the given df parameter. If empty, no values equal to *1* will be added to the new row.
values.neg	a character vector whose elements are indicators of a negative state/condition and will be assigned a value of *-1*. If empty, no values equal to *-1* will be added to the new row. These elements **must be a subset of the column names** of the given df parameter.
pos	string. The position where we should put the new row that will be generated. Two possible values: "first" (default) or "last".
row.name	string. The name of the new row that we will added. Default value: NULL.

Value

the df with one extra row, having elements from the {-1,0,1} set depending on values of values.pos and values.neg vectors.

Examples

```
df = data.frame(c(0,-1,0), c(0,1,-1), c(1,0,0))
colnames(df) = c("A","B","C")
df.new = add_row_to_ternary_df(df, values.pos = c("A"), values.neg = c("C"), row.name = "Hello!")
```

---

add_vector_to_df *Add vector to a (n x 2) data frame*

Description

Given a vector, adds each value and its corresponding name to a data frame of 2 columns as new rows, where the name fills in the 1st column and the value the 2nd column.

Usage

```
add_vector_to_df(df, vec)
```

Arguments

df	data.frame, with n rows and 2 columns
vec	a vector

Value

a data.frame with additional rows and each element as a character.

Examples

```
df = data.frame(c(0,0,1), c(0,0,2))
vec = 1:3
names(vec) = c("a","b","c")

add_vector_to_df(df, vec)
```

---

get_average_over_unique_values

Get average over unique values

Description

Use this function on two vectors with same names attribute (column names), to find for each unique (numeric) value of the first vector, the average and standard deviation values of the second vector's values (matching is done by column name)

Usage

```
get_average_over_unique_values(vec1, vec2)
```

Arguments

vec1	vector with names attribute
vec2	vector with names attribute

Value

A matrix consisting of 3 column vectors. The matrix size is dim(matrix)= 3xn, where n is the number of unique values of vec1). The columns vectors are:

1. the first input vector pruned to its unique values
2. a vector with the average values for each unique value of the first vector (the matching is done by column name)
3. a vector with the standard deviation values for each unique value of the first vector (the matching is done by column name)

Examples

```
vec1 = c(1, 2, 3, 2)
vec2 = c(20, 2, 2.5, 8)
names.vec = c(seq(1,4))
names(vec1) = names.vec
names(vec2) = names.vec

res = get_average_over_unique_values(vec1, vec2)
```

---

get_common_names *Get the common names of two vectors*

Description

This function prints and returns the common `names` of two vectors. The two vectors don't have to be the same length.

Usage

```
get_common_names(vec1, vec2, vector.names.str = "nodes",
  with.gt = TRUE)
```

Arguments

`vec1`	vector with `names` attribute
`vec2`	vector with `names` attribute
`vector.names.str`	string. Used for printing, it tell us what are the `names` of the two vectors (use plural form). Default value: "nodes".
`with.gt`	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.

Value

the character vector of the common names. If there is only one name in common, the `vector.names.str` gets the last character stripped for readability. If there is no common names, it returns FALSE.

See Also

`pretty_print_vector_values`, `pretty_print_string`

Examples

```
vec1 = c(1,1,1)
vec2 = c(1,2)
names(vec1) = c("a","b","c")
names(vec2) = c("c","b")

common.names = get_common_names(vec1, vec2)
```

`get_common_values` *Get the common values of two vectors*

Description

This function prints and returns the common values of two vectors. The two vectors don't have to be the same length.

Usage

```
get_common_values(vec1, vec2, vector.values.str = "nodes",
  with.gt = TRUE)
```

Arguments

`vec1`	vector
`vec2`	vector
`vector.values.str`	string. Used for printing, it tell us what are the values of the two vectors (use plural form). Default value: "nodes".
`with.gt`	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.

Value

the vector of the common values. If there is only one value in common, the `vector.values.str` gets the last character stripped for readability. If there are no common values, it returns NULL.

See Also

`pretty_print_vector_values`, `pretty_print_string`

Examples

```
vec1 = c(1,2,3)
vec2 = c(3,4,1)

common.names = get_common_values(vec1, vec2)
```

get_parent_dir *Retrieve the parent directory*

Description

Use this function to retrieve the parent directory from a string representing the full path of a file or a directory.

Usage

```
get_parent_dir(pathStr)
```

Arguments

pathStr	string. The name of the directory, can be a full path filename.

Value

a string representing the parent directory. When a non-file path is used as input (or something along those lines :) then it returns the root ("/") directory.

Examples

```
get_parent_dir("/home/john")
get_parent_dir("/home/john/a.txt")
get_parent_dir("/home")
```

get_percentage_of_matches *Get percentage of matches between two vectors*

Description

Use this function on two numeric vectors with the same names attribute (columns) and same length, in order to find the percentage of common elements (value matches between the two vectors). The same names for the two vectors ensures that their values are logically matched one-to-one.

Usage

```
get_percentage_of_matches(vec1, vec2)
```

Arguments

vec1	numeric vector with names attribute
vec2	numeric vector with names attribute

Value

the percentage of common values (exact matches) between the two vectors. Can only be a value between 0 (no common elements) and 1 (perfect element match). Note that *NaN* and *NA* values are allowed in the input vectors, but they will always count as a mismatch.

Examples

```
vec1 = c(1, 2, 3, 2)
vec2 = c(20, 2, 2.5, 8)
vec3 = c(1, 2, 333, 222)
names.vec = c(seq(1,4))
names(vec1) = names.vec
names(vec2) = names.vec
names(vec3) = names.vec

match.1.2 = get_percentage_of_matches(vec1, vec2)
match.1.3 = get_percentage_of_matches(vec1, vec3)
```

---

get_ternary_class_id *Get ternary class id*

Description

Helper function that checks if a *value* surpasses the given *threshold* either positively, negatively or not at all and returns a value indicating in which class (i.e. interval) it belongs.

Usage

```
get_ternary_class_id(value, threshold)
```

Arguments

value	numeric
threshold	numeric

Value

an integer. There are 3 cases:

- 1: when $value > threshold$
- -1: when $value < -threshold$
- 0: otherwise

is_between *Is value between two others?*

Description

This function checks if a given value is inside an interval specified by two boundary values.

Usage

```
is_between(value, low.thres, high.thres, include.high.value = FALSE)
```

Arguments

value	numeric
low.thres	numeric. Lower boundary of the interval.
high.thres	numeric. Upper boundary of the interval.
include.high.value	logical. Whether the upper bound is included in the interval or not. Default value: FALSE.

Value

a logical specifying if the value is inside the interval [low.thres,high.thres) (default behaviour) or inside the interval [low.thres,high.thres] if include.high.value is TRUE.

Examples

```
is_between(3,2,4)
is_between(4,2,4)
is_between(4,2,4,include.high.value=TRUE)
```

is_empty *Is object empty?*

Description

A function to test whether an object is **empty**. It checks the length of the object, so it has different behaviour than is.null.

Usage

```
is_empty(obj)
```

Arguments

obj	a general object

Value

a logical specifying if the object is NULL or not.

Examples

```
# TRUE
is_empty(NULL)
is_empty(c())

# FALSE
is_empty("")
is_empty(NA)
is_empty(NaN)
```

---

ldf_arrange_by_rownames

Rearrange a list of data frames by rownames

Description

Rearrange a list of data frames by rownames

Usage

```
ldf_arrange_by_rownames(list_df)
```

Arguments

list_df	a (non-empty) list of data.frame objects. The data frames must have the same colnames attribute.

Value

a rearranged list of data frames, where the names of the elements of the list_df (the 'ids' of the data frames) and the rownames of the data frames have switched places: the unique row names of the original list's combined data frames serve as names for the returned list of data frames, while the data frame 'ids' (names of the original list's elements) now serve as rownames for the data frames in the new list.

E.g. if in the given list there was a data.frame with id 'A': a = list_df[["A"]] and rownames(a) = c("row1","row2"), then in the rearranged list there would be two data frames with ids "row1" and "row2", each of them having a row with name "A" where also these data rows would be the same as before: list_df[["A"]]["row1",] == returned_list[["row1"]]["A",] and list_df[["A"]]["row2",] == returned_list[["row2"]]["A",] respectively.

Examples

```
df.1 = data.frame(matrix(data = 0, nrow = 3, ncol = 3,
  dimnames = list(c("row1", "row2", "row3"), c("C.1", "C.2", "C.3"))))
df.2 = data.frame(matrix(data = 1, nrow = 3, ncol = 3,
  dimnames = list(c("row1", "row2", "row4"), c("C.1", "C.2", "C.3"))))
list_df = list(df.1, df.2)
names(list_df) = c("zeros", "ones")
res_list_df = ldf_arrange_by_rownames(list_df)
```

`make_color_bar_plot` *Make a color bar plot*

Description

Use this function when you want to visualize some numbers and their respective color values. Note that more than 42 colors won't be nice to see (too thin bars)!

Usage

```
make_color_bar_plot(color.vector, number.vector, title,
  x.axis.label = "")
```

Arguments

`color.vector`	vector of color values
`number.vector`	vector of numeric values (same length with `color.vector`)
`title`	string. The title of the barplot
`x.axis.label`	string. The x-axis label. Default value: empty string

Examples

```
color.vector = rainbow(10)
number.vector = 1:10
title = "First 10 rainbow() colors"
make_color_bar_plot(color.vector, number.vector, title)
```

make_multiple_density_plot *Multiple densities plot*

Description

Combine many density distributions to one common plot.

Usage

```
make_multiple_density_plot(densities, legend.title, title, x.axis.label,
  legend.size = 1)
```

Arguments

densities	a list, each element holding the results from executing the density function to a (different) vector. Note that you need to provide a name for each list element for the legend (see example).
legend.title	string. The legend title.
title	string. The plot title.
x.axis.label	string. The x-axis label.
legend.size	numeric. Default value: 1.

Examples

```
mat = matrix(rnorm(60), ncol=20)
densities = apply(mat, 1, density)
names(densities) = c("1st", "2nd", "3rd")
make_multiple_density_plot(densities, legend.title = "Samples",
  x.axis.label = "", title = "3 Normal Distribution Samples")
```

mat_equal *Matrix equality*

Description

Check if two matrices are equal. Equality is defined by both of them being matrices in the first place, having the same dimensions as well as the same elements.

Usage

```
mat_equal(x, y)
```

Arguments

x, y	matrices

Value

a logical specifying if the two matrices are equal or not.

---

normalize_to_range *Range normalization*

Description

Normalize a vector, matrix or data.frame of numeric values in a specified range.

Usage

```
normalize_to_range(x, range = c(0, 1))
```

Arguments

x	vector, matrix or data.frame with at least two different elements
range	vector of two elements specifying the desired normalized range. Default value is c(0,1)

Value

the normalized data

Examples

```
vec = 1:10
normalize_to_range(vec)
normalize_to_range(vec, range = c(-1,1))

mat = matrix(c(0,2,1), ncol = 3, nrow = 4)
normalize_to_range(mat, range = c(-5,5))
```

outersect *Outersect*

Description

Performs set *outersection* on two vectors. The opposite operation from `intersect`!

Usage

```
outersect(x, y)
```

Arguments

`x, y`	vectors

Value

a vector of the non-common elements of x and y.

See Also

`intersect`

Examples

```
x = 1:10
y = 2:11

# c(1,11)
outersect(x,y)
```

plot_string_to_file *Plot string to output format*

Description

Execute a plot string expression and output the result to the specified file format.

Usage

```
plot_string_to_file(file, file.format = c("pdf", "png", "svg", "tiff"),
  plot.string)
```

Arguments

file	string. The name of the file, can be a full path.
file.format	string. The output file format. Can be one of these: pdf, svg, png or tiff.
plot.string	string. The plot string expression.

Examples

```
x = 1:10
y = 1:10
plot_string_to_file(paste0(tempdir(), "/my_file.pdf"), "pdf", "plot(x,y)")
```

---

pretty_print_bold_string

Pretty print a bold string

Description

Prints a bold string only when 'html.output' is enabled. Otherwise, it prints a normal string. The the ">" sign can be appended if nice output in an R notebook is desired.

Usage

```
pretty_print_bold_string(string, with.gt = TRUE, html.output = TRUE)
```

Arguments

string	a string
with.gt	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.
html.output	logical. If TRUE, it encapsulates the string with the bold tags for an HTML document. Default value: TRUE.

See Also

pretty_print_string

pretty_print_name_and_value *Pretty print a name and value*

Description

Pretty print a name and value

Usage

```
pretty_print_name_and_value(name, value, with.gt = FALSE,
  with.comma = TRUE)
```

Arguments

name	string
value	string
with.gt	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: FALSE.
with.comma	logical. Determines if the comma (,) character will be appended to the end of the output. Default value: TRUE.

Examples

```
pretty_print_name_and_value("aName", "aValue", with.gt = TRUE)
pretty_print_name_and_value("aName", "aValue", with.comma = FALSE)
```

pretty_print_string *Pretty print a string*

Description

Nice printing of a string in an R notebook (default behaviour). Otherwise, it prints the string to the standard R output.

Usage

```
pretty_print_string(string, with.gt = TRUE)
```

Arguments

string	a string
with.gt	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.

See Also

cat

---

pretty_print_vector_names
Pretty printing of a vector's names attribute

Description

Pretty printing of a vector's names attribute

Usage

```
pretty_print_vector_names(vec, vector.names.str = "nodes", sep = ", ",
  with.gt = TRUE)
```

Arguments

vec	vector
vector.names.str	string. It tell us what are the names of the vector (use plural form) in order to fill the print message. Default value: "nodes".
sep	string. The separator character to use to distinguish between the names values. Default value: ", ".
with.gt	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.

See Also

pretty_print_string

---

pretty_print_vector_names_and_values
Pretty printing of a vector's names and values

Description

It outputs a vector's names and values in this format: *name1: value1, name2: value2,....* You can choose how many elements to show in this format.

Usage

```
pretty_print_vector_names_and_values(vec, n = -1)
```

Arguments

vec	vector with names attribute
n	the number of elements that you want to print in a nice way. Default value: -1 (pretty print all elements). For any n < 1, all elements are printed.

See Also

pretty_print_name_and_value

---

pretty_print_vector_values

Pretty printing of a vector's values

Description

Pretty printing of a vector's values

Usage

```
pretty_print_vector_values(vec, vector.values.str = "nodes",
  sep = ", ", with.gt = TRUE)
```

Arguments

vec	vector
vector.values.str	string. It tell us what are the values of the vector (use plural form) in order to fill the print message. Default value: "nodes".
sep	string. The separator character to use to distinguish between the vector values. Default value: ", ".
with.gt	logical. Determines if the ">" sign will be appended for nice printing in an R notebook. Default value: TRUE.

See Also

pretty_print_string

print_empty_line *Print an empty line*

Description

Print an empty line

Usage

```
print_empty_line(html.output = FALSE)
```

Arguments

html.output	logical. If TRUE, it outputs an empty line for an HTML document, else an empty line for the standard R output. Default value: FALSE.

See Also

cat

prune_and_reorder_vector *Prune and reorder vector elements*

Description

Given two vectors, the first one's elements are pruned and reordered according to the common values of the second vector and the elements' names *(attribute) of the first*. If there no common such values, an empty vector is returned.

Usage

```
prune_and_reorder_vector(vec, filter.vec)
```

Arguments

vec	a vector with names attribute
filter.vec	a character vector whose values will be used to filter the vec elements

Value

the pruned and re-arranged vector.

Examples

```
vec = c(1,2,3)
names(vec) = c("a","b","c")

filter.vec1 = c("a")
prune_and_reorder_vector(vec, filter.vec1)

filter.vec2 = c("c", "ert", "b")
prune_and_reorder_vector(vec, filter.vec2)
```

---

prune_columns_from_df *Prune single-value columns from a data frame*

Description

Given a `data.frame` and an integer value, it checks whether there is a column vector whose values match the given one. If so, it prunes that single-valued column from the `data.frame`

Usage

```
prune_columns_from_df(df, value)
```

Arguments

`df`	`data.frame`
`value`	an integer value

Value

the column-pruned `data.frame`

Examples

```
df = data.frame(c(0,0,0), c(0,1,0), c(1,0,0))
prune_columns_from_df(df, value = 0)
```

prune_rows_from_df *Prune single-value rows from a data frame*

Description

Given a `data.frame` and an integer value, it checks whether there is a row vector whose values match the given one. If so, it prunes that single-valued row from the `data.frame`

Usage

```
prune_rows_from_df(df, value)
```

Arguments

`df`	`data.frame`
`value`	an integer value

Value

the row-pruned `data.frame`

Examples

```
df = data.frame(c(0,0,0), c(0,1,0), c(1,0,0))
prune_rows_from_df(df, value = 0)
```

remove_commented_and_empty_lines *Remove commented and empty lines*

Description

Removes empty or commented lines from a character vector (each element being a line)

Usage

```
remove_commented_and_empty_lines(lines)
```

Arguments

`lines`	a character vector, usually the result from using the `readLines` function

Value

a character vector of the pruned lines

save_df_to_file *Save data frame to a specified file*

Description

Function for saving a `data.frame` to a specified file.

Usage

```
save_df_to_file(df, file)
```

Arguments

`df`	data.frame
`file`	string. The name of the file, can be a full path.

save_mat_to_file *Save matrix to a specified file*

Description

Function for saving a `matrix` to a specified file. Uses the `save_df_to_file` function.

Usage

```
save_mat_to_file(mat, file)
```

Arguments

`mat`	matrix
`file`	string. The name of the file, can be a full path.

save_vector_to_file *Save vector to a specified file*

Description

Function for saving a `vector` with or without its row names to a specified file.

Usage

```
save_vector_to_file(vector, file, with.row.names = FALSE)
```

Arguments

`vector`	vector
`file`	string. The name of the file, can be a full path.
`with.row.names`	logical. If TRUE, then the `names(vector)` will be included in the output file. Default value: FALSE.

specify_decimal *Specify decimal*

Description

Use this function to transform a given decimal number to the desired precision by choosing the number of digits after the decimal point.

Usage

```
specify_decimal(number, digits.to.keep)
```

Arguments

`number`	numeric
`digits.to.keep`	numeric. Refers to the digits to keep after decimal point '.'. This value should be 15 or less.

Value

the pruned number in string format

Examples

```
# 0.123
specify_decimal(0.1233213, 3)
```

usefun *usefun*

Description

A collection of useful functions by John

Details

For a complete list of functions, use `library(help = "usefun")`

Index