Package ‘usefun’

September 17, 2023

Type Package
Title A Collection of Useful Functions by John
Version 0.5.0
Description A set of general functions that I have used in various projects and other R packages. Miscellaneous operations on data frames, matrices and vectors, ROC and PR statistics.
License MIT + file LICENSE
URL https://github.com/bblodfon/usefun
BugReports https://github.com/bblodfon/usefun/issues
Encoding UTF-8
RoxygenNote 7.2.3
Imports graphics, stats, utils, dplyr, PRROC (>= 1.3.1)
Suggests testthat, covr, readr, mlr3misc
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 2023-09-17 08:40:02 UTC

R topics documented:

add_row_to_ternary_df ........................................... 2
add_vector_to_df .................................................. 3
binarize_to_thres ................................................... 4
colors.100 .......................................................... 5
dec_to_bin ........................................................... 5
get_common_names .................................................... 6
get_common_values ................................................... 7
get_parent_dir ....................................................... 8
get_percentage_of_matches ....................................... 8
**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

```r
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

```r
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)

binarize_to_thres Binarize matrix to given threshold

Description

Simple function that checks every element of a given matrix (or data.frame) if it surpasses the given threshold either positively or negatively and it outputs 1 for that element, otherwise 0.

Usage

binarize_to_thres(mat, thres)

Arguments

mat a matrix or data.frame object
thres a positive numerical value

Value

a binarized matrix (values either 0 or 1): elements that have 1 correspond to values of mat that they were either larger than the threshold or smaller than it’s negative.

Examples

mat = matrix(data = -4:4, nrow = 3, ncol = 3)
binarize_to_thres(mat, thres = 0.5)
binarize_to_thres(mat, thres = 2.5)
colors.100

100 distinct colors

Description

100 as-much-as-possible distinct colors!

Usage

colors.100

Format

An object of class character of length 100.

dec_to_bin

Convert decimal number to its binary representation

Description

Get the binary representation of any decimal number from 0 to \(2^{31} - 1\). Doesn’t work for larger numbers.

Usage

dec_to_bin(decimal_num, bits = 32)

Arguments

decimal_num \hspace{1em} \text{decimal number between 0 and } (2^{31}) - 1

bits \hspace{1em} \text{number of bits to keep in the result counting from the right. Default value is 32.}

Value

a binary string representation of the given decimal number.

Examples

# representing 0
dec_to_bin(0,1)
dec_to_bin(0,10)
dec_to_bin(0,32)
dec_to_bin(0)

# representing 24
dec_to_bin(24,6)
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 (use with the chuck option results = 'asis'). 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

```r
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 tells 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 (use with the chunk option `results = 'asis'`). 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

```r
vec1 = c(1,2,3)
vec2 = c(3,4,1)
common.names = get_common_values(vec1, vec2)
```
get_parent_dir  
Retrieves 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
get_roc_stats

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(vec1) = c(seq(1,4))
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_roc_stats
Generate ROC statistics

Description
Use this function to generate the most useful statistics related to the generation of a basic ROC (Receiver Operating Characteristic) curve.

Usage
get_roc_stats(df, pred_col, label_col, direction = "<")

Arguments
df a data.frame with (at least) two columns. See next two parameters for what values these two columns should have (which should match one to one).
pred_col string. The name of the column of the df data.frame that has the prediction values. The values can be any numeric, negative, positive or zero. What matters is the ranking of these values which is clarified with the direction parameter.
label_col string. The name of the column of the df data.frame that has the true positive labelings/observed classes for the prediction values. This column must have either 1 or 0 elements representing either a positive or negative classification label for the corresponding values.
direction string. Can be either > or < (default value) and indicates the direction/ranking of the prediction values with respect to the positive class labeling (for a specific threshold). If smaller prediction values indicate the positive class labeling (e.g. probability of positive class), use < whereas if larger prediction values indicate the positive class labeling (e.g. probability of positive class), use >.
get_roc_stats

Value

A list with two elements:

- **roc_stats**: a tibble which includes the thresholds for the ROC curve and the confusion matrix stats for each threshold as follows: **TP** (#True Positives), **FN** (#False Negatives), **TN** (#True Negatives), **FP** (#False Positives), **FPR** (False Positive Rate - the x-axis values for the ROC curve) and **TPR** (True Positive Rate - the y-axis values for the ROC curve). Also included are the dist-from-chance (the vertical distance of the corresponding (FPR,TPR) point to the chance line or positive diagonal) and the dist-from-0-1 (the euclidean distance of the corresponding (FPR,TPR) point from (0,1)).

- **AUC**: a number representing the Area Under the (ROC) Curve.

The returned results provide an easy way to compute two optimal cutpoints (thresholds) that dichotomize the predictions to positive and negative. The first is the **Youden index**, which is the maximum vertical distance from the ROC curve to the chance line or positive diagonal. The second is the point of the ROC curve closest to the (0,1) - the point of perfect differentiation. See examples below.

Examples

```r
# load libraries
library(readr)
library(dplyr)

# load test tibble
test_file = system.file("extdata", "test_df.tsv", package = "usefun", mustWork = TRUE)
test_df = readr::read_tsv(test_file, col_types = "di")

# get ROC stats
res = get_roc_stats(df = test_df, pred_col = "score", label_col = "observed")

# Plot ROC with a legend showing the AUC value
plot(x = res$roc_stats$FPR, y = res$roc_stats$TPR,
     type = 'l', lwd = 2, col = '#377EB8', main = 'ROC curve',
     xlab = 'False Positive Rate (FPR)', ylab = 'True Positive Rate (TPR)'
)
legend('bottomright', legend = round(res$AUC, digits = 3),
       title = 'AUC', col = '#377EB8', pch = 19)
grid()
abline(a = 0, b = 1, col = '#FF726F', lty = 2)

# Get two possible cutoffs
youden_index_df = res$roc_stats %>%
  filter(dist_from_chance == max(dist_from_chance))
min_classification_df = res$roc_stats %>%
  filter(dist_from_0_1 == min(dist_from_0_1))
```
**get_stats_for_unique_values**

*Get stats for 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**

```r
get_stats_for_unique_values(vec1, vec2)
```

**Arguments**

- `vec1`: vector with names attribute
- `vec2`: vector with names attribute

**Value**

A `data.frame` consisting of 3 column vectors. The `data.frame` size is nx3, where n is the number of unique values of `vec1` (rows). 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**

```r
gen1 = 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_stats_for_unique_values(vec1, vec2)
```
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 \texttt{value > threshold}
• -1: when \texttt{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.
is_empty

Value

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

Examples

\[
\begin{align*}
is\cdot between(3, 2, 4) \\
is\cdot between(4, 2, 4) \\
is\cdot between(4, 2, 4, include\cdot high\cdot value=TRUE)
\end{align*}
\]

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 \texttt{is.null}.

Usage

\[
is\cdot empty(obj)
\]

Arguments

\[
obj \quad \text{a general object}
\]

Value

a logical specifying if the object is NULL or not.

Examples

\[
\begin{align*}
\# \ TRUE \\
is\cdot empty(NULL) \\
is\cdot empty(c())
\end{align*}
\]

\[
\begin{align*}
\# \ FALSE \\
is\cdot empty(""") \\
is\cdot empty(NA) \\
is\cdot empty(NaN)
\end{align*}
\]
Rearrange a list of data frames by rownames

## Description

Rearrange a list of data frames by rownames

## Usage

```r
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':

```r
a = list_df[['A']]
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:

```r
list_df[['A']][["row1", ]] == returned_list[['row1']]["A", ] and list_df[['A']][["row2", ]] == returned_list[['row2']]["A", ] respectively.
```

## Examples

```r
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**

```r
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**

```r
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**

```r
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**
\[
\text{normalize\_to\_range}(x, \text{range} = c(0, 1))
\]

**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**
\[
\text{vec} = 1:10 \\
\text{normalize\_to\_range}(\text{vec}) \\
\text{normalize\_to\_range}(\text{vec}, \text{range} = c(-1,1))
\]
\[
\text{mat} = \text{matrix}(c(0,2,1), \text{ncol} = 3, \text{nrow} = 4) \\
\text{normalize\_to\_range}(\text{mat}, \text{range} = c(-5,5))
\]

**outersect**  
*Outersect*

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

**Usage**
\[
\text{outersect}(x, y)
\]

**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)

Description

Get partial permutation of a vector

Usage

partial_permut(x, exp_sim = 0)

Arguments

x a vector with at least 2 elements
exp_sim a value between 0 and 1 indicating the level of expected similarity between the input and output vector. Default value is 0 (random permutation).

Value

a partially (random) permutated vector. If exp_sim = 0 then the result is equal to sample(x) (a random permutation). If exp_sim = 1 then the result is always the same as the input vector. For exp_sim values between 0 and 1 we randomly sample a subset of the input vector inversely proportionate to the exp_sim value (e.g. exp_sim = 0.8 => 20% of the elements) and randomly permutate these elements only.

Examples

set.seed(42)
partial_permut(x = LETTERS, exp_sim = 0)
partial_permut(x = LETTERS, exp_sim = 0.5)
partial_permut(x = LETTERS, exp_sim = 0.9)
pr.test

Compare two Precision-Recall curves

Description

Test the hypothesis that the true difference in PR AUCs is equal to 0. We implement the same bootstrap method based on the idea from \texttt{pROC::roc.test()}. The PR AUC is calculated using \texttt{PRROC::pr.curve()} with the interpolation method of Davis (2006).

Usage

\begin{verbatim}
pr.test(
  labels,
  pred1,
  pred2,
  boot.n = 10000,
  boot.stratified = TRUE,
  alternative = "two.sided"
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{labels} \hspace{1cm} \texttt{numeric()}
    Vector of responses/labels (only two classes/values allowed: cases/positive class = 1 and controls/negative class = 0)
  \item \texttt{pred1} \hspace{1cm} \texttt{numeric()}
    Vector of prediction values. Higher values denote positive class.
  \item \texttt{pred2} \hspace{1cm} \texttt{numeric()}
    Vector of prediction values. Higher values denote positive class. Must have the same length as \texttt{pred1}.
  \item \texttt{boot.n} \hspace{1cm} \texttt{numeric(1)}
    Number of bootstrap resamples. Default: 10000
  \item \texttt{boot.stratified} \hspace{1cm} \texttt{logical(1)}
    Whether the bootstrap resampling is stratified (same number of cases/controls in each replicate as in the original sample) or not. Advised to use especially when classes from \texttt{labels} are imbalanced. Default: TRUE.
  \item \texttt{alternative} \hspace{1cm} \texttt{character(1)}
    Specifies the alternative hypothesis. Either "two.sided", "less" or "greater". Default: "two.sided".
\end{itemize}

Value

a list with the AUCs of the two original prediction vectors and the p-value of the bootstrap-based test.
References


Examples

```r
set.seed(42)
# imbalanced labels
labels = sample(c(0,1), 20, replace = TRUE, prob = c(0.8,0.2))
# predictions
pred1 = rnorm(20)
pred2 = rnorm(20)
pr.test(labels, pred1, pred2, boot.n = 1000, boot.stratified = FALSE)
pr.test(labels, pred1, pred2, boot.n = 1000, boot.stratified = TRUE)
```

---

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

```r
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. (use with the chuck option `results = 'asis'`). 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

**Description**

Pretty print a name and value

**Usage**

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

**Arguments**

- `name`: string
- `value`: string
- `with.gt`: logical. Determines if the "\(\geq\)" sign will be appended for nice printing in an R notebook (use with the `chuck` option `results = 'asis'`). Default value: FALSE.
- `with.comma`: logical. Determines if the comma (,) character will be appended to the end of the output. Default value: TRUE.

**Examples**

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

### pretty_print_string

**Description**

Pretty print a string

**Usage**

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

**Arguments**

- `string`: a string
- `with.gt`: logical. Determines if the "\(\geq\)" sign will be appended for nice printing in an R notebook (use with the `chuck` option `results = 'asis'`). 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 = ", "
)

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 (use with the chuck option results = ‘asis’). 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. Use with the chuck option results = ‘asis’ to get
a nice printing in an R notebook.

Usage

pretty_print_vector_names_and_values(vec, n = -1)
pretty_print_vector_values

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

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 (use with the chuck option results = 'asis'). Default value: TRUE.

See Also

pretty_print_string
print_empty_line | **Print an empty line**
---

**Description**

Print an empty line

**Usage**

```r
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**

```r
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.
prune_columns_from_df

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)
remove_commented_and_empty_lines

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

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)
save_df_to_file  Save data frame to a specified file

Description
Function for saving a data.frame to a specified file. Column and row names are written by default and the tab is used as a delimiter.

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. By default the tab is used as a delimiter.

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)
Description

A collection of useful functions by John

Details

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

* datasets
  colors.100, 5

add_row_to_ternary_df, 2
add_vector_to_df, 3
binarize_to_thres, 4
cat, 21, 24
colors.100, 5
dec_to_bin, 5
density, 16
get_common_names, 6
get_common_values, 7
get_parent_dir, 8
get_percentage_of_matches, 8
get_roc_stats, 9
get_stats_for_unique_values, 11
get_ternary_class_id, 12
intersect, 18
is.null, 13
is_between, 12
is_empty, 13
ldf_arrange_by_rownames, 14
make_color_bar_plot, 15
make_multiple_density_plot, 15
mat_equal, 16
normalize_to_range, 17
outersect, 17
partial_permut, 18
pr.test, 19
pretty_print_bold_string, 20
pretty_print_name_and_value, 21, 23
pretty_print_string, 6, 7, 20, 21, 22, 23
pretty_print_vector_names, 22
pretty_print_vector_names_and_values, 22
pretty_print_vector_values, 6, 7, 23
print_empty_line, 24
pROC::roc.test(), 19
PRROC::pr.curve(), 19
prune_and_reorder_vector, 24
prune_columns_from_df, 25
prune_rows_from_df, 26
readLines, 26
remove_commented_and_empty_lines, 26
save_df_to_file, 27, 27
save_mat_to_file, 27
save_vector_to_file, 28
specify_decimal, 28
usefun, 29
usefun-package (usefun), 29