Package 'hutils'

May 12, 2018

Type Package

Title Miscellaneous R Functions and Aliases

Version 1.1.0

Date 2018-05-13

Maintainer Hugh Parsonage <hugh.parsonage@gmail.com>

Description Provides utility functions for, and drawing on, the 'data.table' package. The package also collates useful miscellaneous functions extending base R not available elsewhere. The name is a portmanteau of 'utils' and the author.

BugReports https://github.com/hughparsonage/hutils/issues

URL https://github.com/hughparsonage/hutils

License GPL-3

Depends R (>= 3.1.0)

Imports data.table (< 2.0.0), magrittr (< 2.0.0), stats (< 4.0.0), utils (< 4.0.0), fastmatch (< 2.0.0), grDevices (< 4.0.0)

Suggests testthat, datasets, dplyr, microbenchmark, knitr, rmarkdown, nycflights13, geosphere, ggplot2

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Hugh Parsonage [aut, cre], Michael Frasco [ctb], Ben Hamner [ctb]

Repository CRAN

Date/Publication 2018-05-12 14:58:06 UTC

R topics documented:

`hutils-package` *hutils package*

Description

Provides utility functions for, and drawing on, the 'data.table' package. The package also collates useful miscellaneous functions extending base R not available elsewhere. The name is a portmanteau of 'utils' and the author.

aliases *Aliases*

Description

These simple aliases can be useful to avoid operator precedence ambiguity, or to make use of indents from commas within your text editor. The all-caps versions accept single-length (capable of 'short-circuits') logical conditions only.

Neithers and nors are identical except have slightly different short-circuits. `NOR` uses negation once so may be quicker if the first argument is very, very prompt.

Usage

```
AND(x, y)

OR(x, y)

nor(x, y)

neither(x, y)

NOR(x, y)

NEITHER(x, y)

pow()
```

Arguments

`x, y`	Logical conditions.

auc *AUC*

Description

Returns the area under the curve ("AUC") of a receiver-operating characteristic curve for the given predicted and actual values.

Usage

```
auc(actual, pred)
```

Arguments

`actual`	Logical vector: `TRUE` for positive class. If not a logical vector, the result is interpreted as one if safe to do so, *viz.* if `actual` contains precisely two unique values and is either a numeric vector, an ordered factor, or the unique values are `FALSE` and `TRUE` (case-insensitively). Anything else is an error.
`pred`	Numeric (double) vector the same length as `actual` giving the predicted probability of `TRUE`. Must be a numeric vector the same length as `actual`.

Author(s)

Source

Source code based on `Metrics::auc` from Ben Hamner and Michael Frasco and Erin LeDell from the Metrics package.

`coalesce` *Find first non-missing element*

Description

Lightweight version of `dplyr::coalesce`, with all the vices and virtues that come from such an approach. Very similar logic (and timings to `dplyr::coalesce`), though no ability to use quosures etc. One exception is that if `x` does not contain any missing values, it is returned immediately, and ignores `...`. For example, `dplyr::coalesce(1:2, 1:3)` is an error, but `hutils::coalesce(1:2, 1:3)` is not.

Usage

```
coalesce(x, ...)
```

Arguments

x	A vector
...	Successive vectors whose values will replace the corresponding values in x if the value is (still) missing.

Value

x with missing values replaced by the first non-missing corresponding elements in .... That is, if ... = A, B, C and x[i] is missing, then x[i] is replaced by A[i]. If x[i] is still missing (i.e. A[i] was itself NA), then it is replaced by B[i], C[i] until it is no longer missing or the list has been exhausted.

Source

Original source code but obviously inspired by dplyr::coalesce.

Examples

```
coalesce(c(1, NA, NA, 4), c(1, 2, NA, NA), c(3, 4, 5, NA))
```

---

dev_copy2a4 *Copy device to an A4 PDF*

Description

Simply a wrapper around dev.copy2pdf, but without the need to remember that an A4 sheet of paper is 8.27 in by 11.69 in.

Usage

```
dev_copy2a4(filename, ...)
```

Arguments

filename	A string giving the name of the PDF file to write to, must end in .pdf.
...	Other parameters passed to pdf.

Value

As in dev2.

drop_col *Drop column or columns*

Description

Drop column or columns

Usage

```
drop_col(DT, var, checkDT = TRUE)

drop_cols(DT, vars, checkDT = TRUE)
```

Arguments

DT	A data.table.
var	Quoted column to drop.
checkDT	Should the function check DT is a data.table?
vars	Character vector of columns to drop. Only the intersection is dropped; if any vars are not in names(DT), no warning is emitted.

Value

DT with specified columns removed.

Examples

```
if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)
  DT <- data.table(x = 1, y = 2, z = 3)

  drop_col(DT, "x")
}
```

drop_colr *Drop columns whose names match a pattern*

Description

Drop columns whose names match a pattern

Usage

```
drop_colr(DT, pattern, ..., checkDT = TRUE)
```

Arguments

DT	A data.table.
pattern	A regular expression as in grepl.
...	Arguments passed to grepl.
checkDT	If TRUE (the default), will error if DT is not a data.table.

---

drop_constant_cols *Drop constant columns*

Description

Drops columns that have only one value in a data.table.

Usage

```
drop_constant_cols(DT, copy = FALSE)
```

Arguments

DT	A data.table.
copy	(logical, default: FALSE) Whether the data.table should be copied before any columns are dropped. If FALSE, the default, columns are dropped from DT by reference.

Details

If DT is a data.frame that is not a data.table, constant columns are still dropped, but since DT will be copied, copy should be set to TRUE to avoid a warning. If DT is a data.frame and all but one of the columns are constant, a data.frame will still be returned, as opposed to the values of the sole remaining column, which is the default behaviour of base data.frame.

If all columns are constant, drop_constant_cols returns a Null data table if DT is a data.table, but a data frame with 0 columns and nrow(DT) otherwise.

Examples

```
library(data.table)
X <- data.table(x = c(1, 1), y = c(1, 2))
drop_constant_cols(X)
```

drop_empty_cols *Drop empty columns*

Description

Removes columns from a data.table where all the values are missing.

Usage

```
drop_empty_cols(DT, copy = FALSE)
```

Arguments

DT	A data.table.
copy	Copies the data.table so the original can be retained. Not applicable if DT is not a data.table. If FALSE, the default, DT itself will be modified.

duplicated_rows *Return duplicated rows of data.table*

Description

This function differs from duplicated in that it returns both the duplicate row and the row which has been duplicated. This may prove useful in combination with the by argument for determining whether two observations are identical across more than just the specified columns.

Usage

```
duplicated_rows(DT, by = names(DT), na.rm = FALSE, order = TRUE,
  copyDT = TRUE, na.last = FALSE)
```

Arguments

DT	A data.table.
by	Character vector of columns to evaluate duplicates over.
na.rm	(logical) Should NAs in by be removed before returning duplicates? (Default FALSE.)
order	(logical) Should the result be ordered so that duplicate rows are adjacent? (Default TRUE.)
copyDT	(logical) Should DT be copied prior to detecting duplicates. If FALSE, the ordering of DT will be changed by reference.
na.last	(logical) If order is TRUE, should NAs be ordered first or last?. Passed to data.table::setorderv.

Value

Duplicate rows of DT by by. For interactive use.

Examples

```
if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)

  DT <- data.table(x = rep(1:4, 3),
                   y = rep(1:2, 6),
                   z = rep(1:3, 4))

  # No duplicates
  duplicated_rows(DT)

  # x and y have duplicates
  duplicated_rows(DT, by = c("x", "y"), order = FALSE)

  # By default, the duplicate rows are presented adjacent to each other.
  duplicated_rows(DT, by = c("x", "y"))
}
```

---

find_pattern_in *Find string pattern in (text) file*

Description

Find string pattern in (text) file

Usage

```
find_pattern_in(file_contents, basedir = ".", dir_recursive = TRUE,
  reader = readLines, include.comments = FALSE, use.OS = FALSE,
  file_pattern = "\\.(R|r)(nw|md)?$", file_contents_perl = TRUE,
  file_contents_fixed = FALSE, file.ext = NULL)
```

Arguments

file_contents	A perl-regular expression as a search query.
basedir	The root of the directory tree in which files will be searched recursively.
dir_recursive	(logical, default: TRUE) Search within subdirectories of basedir?
reader	A function, akin to base::readLines, the default, that accepts a filename and returns a character vector.
include.comments	If FALSE, the default, comments (i.e. anything after a #) are not searched.

use.OS	Use the operating system to determine file list. Only available on Windows. If it fails, a fall-back option (using dir) is used.
file_pattern	A regular expression passed to list.files(pattern = file.ext). By default, "\.(R|r)(nw|md)?$", i.e. all R and Sweave files. (Does not have to be a file extension.)
file_contents_perl	(logical, default: TRUE) Should file_contents be interpreted as a perl regex?
file_contents_fixed	(logical, default: FALSE) Should file_contents be interpreted as a fixed regex?
file.ext	A file extension passed to the operating system if use.OS is used.

Value

A data.table, one row per filename with a match, which includes the first line that matched.

`generate_LaTeX_manual` *Generate LaTeX manual of installed package*

Description

Generate LaTeX manual of installed package

Usage

```
generate_LaTeX_manual(pkg, launch = TRUE)
```

Arguments

pkg	Quoted package name (must be installed).
launch	Should the PDF created be launched using the viewer (TRUE by default)?

Value

See system. Called for its side-effect: creates a PDF in the current working directory. Requires a TeX distribution.

Source

https://stackoverflow.com/a/30608000/1664978

haversine_distance *Distance between two points on the Earth*

Description

Distance between two points on the Earth

Usage

```
haversine_distance(lat1, lon1, lat2, lon2)
```

Arguments

lat1, lon1, lat2, lon2	That latitudes and longitudes of the two points.

Details

This is reasonably accurate for distances in the order of 1 to 1000 km.

Value

The distance in kilometres between the two points.

Examples

```
# Distance from YMEL to YSSY
haversine_distance(-37 - 40/60, 144 + 50/60, -33 - 56/60, 151 + 10/60)
```

if_else *Vectorized if*

Description

Lightweight `dplyr::if_else` with the virtues and vices that come from such an approach. Attempts to replicate `dplyr::if_else` but written in base R for faster compile time. `hutils::if_else` should be faster than `dplyr::if_else` ... when it works, but will not work on lists or on factors. Additional attributes may be dropped.

Usage

```
if_else(condition, true, false, missing = NULL)
```

Arguments

condition	Logical vector
true, false	Where condition is TRUE/FALSE, use the correspondingtrue/no value. Must have the same type as each other and be the same length as condition or length-one.
missing	If condition is NA, use the corresponding na value. Liketrue andfalse, must be of the same type and have the same length as condition, unless it has length one.

Value

Where condition is TRUE, the corresponding value in true; where condition is FALSE, the corresponding value in false. Where condition is NA, then the corresponding value in na – unless na is NULL (the default) in which case the value will be NA (with the same type as true.)

Source

Original code but obviously heavily inspired by https://CRAN.R-project.org/package=dplyr.

---

implies *#' Logical implies*

Description

Returns the result of $x \implies y$.

Usage

```
implies(x, y)

x %implies% y
```

Arguments

x, y	Logical vectors of the same length.

Value

Logical implies: TRUE unless x is TRUE and y is FALSE.

NA in either x or y results in NA if and only if the result is unknown. In particular NA %implies% TRUE is TRUE and FALSE %implies% NA is TRUE.

If x or y are length-one, the function proceeds as if the length-one vector were recycled to the length of the other.

Examples

```
library(data.table)
CJ(x = c(TRUE,
         FALSE),
   y = c(TRUE,
         FALSE))[, ` x => y` := x %implies% y][]

#>        x     y  x => y
#> 1: FALSE FALSE    TRUE
#> 2: FALSE  TRUE    TRUE
#> 3:  TRUE FALSE   FALSE
#> 4:  TRUE  TRUE    TRUE

# NA results:
#> 5:    NA    NA      NA
#> 6:    NA FALSE      NA
#> 7:    NA  TRUE    TRUE
#> 8: FALSE    NA    TRUE
#> 9:  TRUE    NA      NA
```

`isTrueFalse` *Logical assertions*

Description

Logical assertions

Usage

```
isTrueFalse(x)
```

Arguments

`x`	An object whose values are to be checked.

Value

For `isTrueFalse`, `TRUE` if and only if `x` is `TRUE` or `FALSE` identically (perhaps with attributes).

mutate_other *Group infrequent entries into 'Other category'*

Description

Useful when you want to constrain the number of unique values in a column by keeping only the most common values.

Usage

```
mutate_other(.data, var, n = 5, count, by = NULL, var.weight = NULL,
  mass = NULL, copy = TRUE, other.category = "Other")
```

Arguments

.data	Data containing variable.
var	Variable containing infrequent entries, to be collapsed into "Other".
n	Threshold for total number of categories above "Other".
count	Threshold for total count of observations before "Other".
by	Extra variables to group by when calculating n or count.
var.weight	Variable to act as a weight: var's where the sum of this variable exceeds mass will be kept, others set to other.category.
mass	Threshold for sum of var.weight: any var where the aggregated sum of var.weight exceeds mass will be kept and other var will be set to other.category. By default (mass = NULL), the value of mass is $-\infty$, with a warning. You may set it explicitly to -Inf if you really want to avoid a warning that this function will have no effect.
copy	Should .data be copied? Currently only TRUE is supported.
other.category	Value that infrequent entries are to be collapsed into. Defaults to "Other".

Value

.data but with var changed so that infrequent values have the same value (other.category).

Examples

```
library(data.table)
library(magrittr)

DT <- data.table(City = c("A", "A", "B", "B", "C", "D"),
                 value = c(1, 9, 4, 4, 5, 11))

DT %>%
  mutate_other("City", var.weight = "value", mass = 10) %>%
  .[]
```

ngrep *Anti-grep*

Description

It is not simple to negate a regular expression. This obviates the need takes the long way round: negating the corresponding `grepl` call.

Usage

```
ngrep(pattern, x, value = FALSE, ...)
```

Arguments

`x, value, pattern`	As in `grep`.
`...`	Arguments passed to `grepl`.

Value

If `value` is `FALSE` (the default), indices of `x` which do not match the pattern; if `TRUE`, the values of `x` themselves.

Examples

```
 grep("[a-h]", letters)
ngrep("[a-h]", letters)

txt <- c("The", "licenses", "for", "most", "software", "are",
"designed", "to", "take", "away", "your", "freedom",
"to", "share", "and", "change", "it.",
"", "By", "contrast,", "the", "GNU", "General", "Public", "License",
"is", "intended", "to", "guarantee", "your", "freedom", "to",
"share", "and", "change", "free", "software", "--",
"to", "make", "sure", "the", "software", "is",
"free", "for", "all", "its", "users")

 grep("[gu]", txt, value = TRUE)
ngrep("[gu]", txt, value = TRUE)
```

provide.dir *Provide directory*

Description

Provide directory. Create directory only if it does not exist.

Usage

```
provide.dir(path, ...)
```

Arguments

path	Path to create.
...	Passed to dir.create.

report_error *Report errors and warnings*

Description

Provides a consistent style for errors and warnings.

Usage

```
report_error(faulty_input, error_condition, requirement, context = NULL,
  advice, hint = NULL, halt = TRUE)
```

Arguments

faulty_input	Unquoted function argument that is the cause of the error condition.
error_condition	A sentence explaining the condition that invoked the error.
requirement	A sentence that explains what is required.
context	(Optional) A sentence that contextualizes the error
advice	Advice for the user to avoid the error.
hint	If the input can be guessed,
halt	(logical, default: TRUE) Should the function signal an error and halt?

`select_grep` *Select names matching a pattern*

Description

Select names matching a pattern

Usage

```
select_grep(DT, patterns, .and = NULL, .but.not = NULL,
  ignore.case = FALSE, perl = TRUE, fixed = FALSE, useBytes = FALSE,
  invert = FALSE, .warn.fixed.mismatch = TRUE)
```

Arguments

`DT`	A `data.frame`.
`patterns`	Regular expressions to be matched against the names of `DT`. If `length(patterns) > 1` the patterns are concatenated using alternation.
`.and`	Character or integer positions of names to select, regardless of whether or not they are matched by `patterns`.
`.but.not`	Character or integer positions of names to drop, regardless of whether or not they are matched by `patterns` or whether they are explicitly added by `.and`.
`ignore.case, perl, fixed, useBytes, invert`	Arguments passed to grep. Note that `perl = TRUE` by default (unlike `grep`) unless `fixed = TRUE` (and `perl` is missing).
`.warn.fixed.mismatch`	(logical, default: `TRUE`) If `TRUE`, the default, selecting `fixed = TRUE` with `perl = TRUE` or `ignore.case = TRUE` results in `perl` and `ignore.case` being reset to `FALSE` with a warning (as in `grep`), even if it makes no difference to the columns eventually selected. If `FALSE` unambiguous results are allowed; if `ignore.case = TRUE` and `fixed = TRUE`, the result is **unambiguous** if `select_grep(DT, tolower(patterns), fixed = TRUE)` and `select_grep(DT, toupper(patterns), fixed = TRUE)` are identical.

Value

`DT` with the selected names.

integer vector of positions

Examples

```
library(data.table)
dt <- data.table(x1 = 1, x2 = 2, y = 0)
select_grep(dt, "x")
select_grep(dt, "x", .and = "y")
select_grep(dt, "x", .and = "y", .but.not = "x2")
```

select_which *Select columns satisfying a condition*

Description

Select columns satisfying a condition

Usage

```
select_which(DT, Which, .and.dots = NULL, checkDT = TRUE)
```

Arguments

DT	A data.table.
Which	A function that takes a vector and returns TRUE or FALSE. TRUE columns are selected.
.and.dots	Optional extra columns to include. May be a character vector of names(DT) or numeric (positions) or logical. If provided, the columns so added (if they do not satisfy Which) will be after all the columns Which do so satisfy.
checkDT	If TRUE (the default), an informative error message is provided if DT is not a data.table.

Value

DT with the selected variables.

Examples

```
library(data.table)
DT <- data.table(x = 1:5,
                 y = letters[1:5],
                 AB = c(NA, TRUE, FALSE))
select_which(DT, anyNA, .and.dots = "y")
```

set_cols_first *Put columns first or last*

Description

Reorder columns of a data.table (via setcolorder) so that particular columns appear first (or last), or in a particular order.

Usage

```
set_cols_first(DT, cols, intersection = TRUE)

set_cols_last(DT, cols, intersection = TRUE)

set_colsuborder(DT, cols, intersection = TRUE)
```

Arguments

DT	A data.table.
cols	Character vector of columns to put before (after) all others or, in the case of set_colsuborder, a vector of columns in the order requested.
intersection	Use the intersection of the names of DT and cols. If FALSE any cols are not the names of DT, the function may error on behalf of data.table. Not available for set_colsuborder.

Details

In the case of set_colsuborder the group of columns cols occupy the same positions in DT but in a different order. See examples.

Examples

```
library(data.table)

DT <- data.table(y = 1:5, z = 11:15, x = letters[1:5])
set_cols_first(DT, "x")[]
set_cols_last(DT, "x")[]
set_colsuborder(DT, c("x", "y"))[]
```

---

unique-keys *Unique keys*

Description

A data.table's key need not be unique, but there are frequently circumstances where non-unique keys can wreak havoc. has_unique_key reports the existence of a unique key, and set_unique_key both sets and ensures the uniqueness of keys.

Usage

```
has_unique_key(DT)

set_unique_key(DT, ...)
```

Arguments

DT	A data.table
...	keys to set

Value

has_unique_key returns TRUE if DT has a unique key, FALSE otherwise. set_unique_key runs setkey(DT, ...) then checks whether the key is unique, returning the keyed data.table if the key is unique, or an error message otherwise.

---

weight2rows *Expand a weighted data frame to an equivalent unweighted*

Description

Expand a weighted data frame to an equivalent unweighted

Usage

```
weight2rows(DT, weight.var)
```

Arguments

DT	A data.table. Will be converted to one if possible.
weight.var	Variable in DT to be used as weights.

Value

DT but with the number of rows expanded to sum(DT[[weight.var]]) to reflect the weighting.

Examples

```
library(data.table)
DT <- data.table(x = 1:5, y = c(1, 1, 1, 1, 2))
weight2rows(DT, "y")
```

%ein% *Exists and (not) in*

Description

A common blunder in R programming is to mistype one of a set of filters without realizing. This function will error if any member of the values to be matched against is not present.

Usage

```
lhs %ein% rhs

lhs %enotin% rhs
```

Arguments

lhs	Values to be matched
rhs	Values to be matched against.

Value

Same as %in% and %notin%, unless an element of rhs is not present in lhs, in which case, an error.

Examples

```
# Incorrectly assumed to include two Species
iris[iris$Species %in% c("setosa", "versicolour"), ]
## Not run:
# Error:
iris[iris$Species %ein% c("setosa", "versicolour"), ]

## End(Not run)
```

%notchin% *Negation of in (character)*

Description

Negation of in (character)

Usage

```
x %notchin% y
```

Arguments

x	Values to be matched.
y	Values to be matched against.

Details

If y is NULL, then x is TRUE for consistency with %in%. If x and y are not both character, the function simply falls back to %in% rather than erroring.

%notin% *Negation of in*

Description

Negation of in

Usage

```
x %notin% y
```

Arguments

x	Values to be matched
y	Values to be matched against.

Details

If y is NULL, then x is TRUE for consistency with %in%. Note that the function uses fmatch internally for performance on large y. Accordingly, y will be modified by adding a .match.hash attribute and thus must not be used in packages where y is a constant, or for things like names of data.table.

%pin% *Partial in*

Description

Analogue of %in% but indicating partial match of the left operand.

Usage

```
x %pin% Y
```

Arguments

x	The values to be matched. Same as %in%.
Y	A vector of values (perl regular expressions) to be matched against.

Value

TRUE for every x for which any grepl is TRUE.

Examples

```
x <- c("Sydney Airport", "Melbourne Airport")

x %pin% c("Syd", "Melb")
```

Index