Package 'rccmisc'

December 6, 2016

Type Package

Title Miscellaneous R Functions for Swedish Regional Cancer Centers

Version 0.3.7

Date 2016-06-08

Author Erik Bulow

Maintainer Erik Bulow <erik.bulow@rccvast.se>

Description Functions either required by other Swedish Regional Cancer Center packages or standalone functions outside the scope of other packages.

License GPL-2

Imports dplyr

Suggests checkpoint, testthat, R.rsp, rvest, xml2

BugReports https://bitbucket.com/cancercentrum/rccmisc/issues

LazyLoad TRUE

VignetteBuilder R.rsp

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-12-06 11:28:47

R topics documented:

`as_numeric` *Test object for, or coerce to, numeric*

Description

`as_numeric` is essentially a wrapper to `as.numeric` except that objects of class factor are first coerced to character and then to numeric. `is_numeric` test if `x` is "somehow numeric" (see examples).

Usage

```
as_numeric(x)

is_numeric(x)
```

Arguments

`x`	object to be coerced or tested (and return a logical vector of the same length) or should it test the whole vector as one object and return a logical vector of length one. (`TRUE` by default).

Examples

```
df <- data.frame(v = c("46513", "45"))
class(df$v) # factor

# Note that
as.numeric(df$v) # 2 1
# but
as_numeric(df$v) # 46513    45

is_numeric(1) # TRUE
is_numeric("1") # TRUE
is_numeric(as.factor(1)) # TRUE
is_numeric(as.factor("khb")) # FALSE
```

best_match *Tries to correct misspelling of character string*

Description

This function uses fuzzy string matching to replace one possibly misspeled (or in other way not fully correct) character string with a correct version of the same string.

Usage

```
best_match(x, key, no_match = NA, all = FALSE)
```

Arguments

x	is a character string (or a character vector) that should be matched to the key
key	is a vector containging the correct spellings of the character strings.
no_match	Output value if there is no match. Default is NA. The input is returned unchanged if not matched and no_match = NULL.
all	is a boolean indicator to specify what happens if there is more than one match. Default is FALSE resulting in a warning message and that only the first match is used. If TRUE the returned vector will no longer have the same length as x.

Value

The function returns a character vector of the same length as x if all = FALSE but with each element substituted to its best match in the key-vector. Strings that could not be matched are NA if (no_match = TRUE) or unchanged if no_match = FALSE. If all = TRUE, one input character string could result in more than one output character string. The output might therefore be longer than the input.

See Also

clean_text

Examples

```
best_match(c("Hej_apa!", "erik", "babian"), c("hej apa", "hej bepa", "kungen", "Erik"))
best_match(c("Hej_apa", "erik", "babian"),
   c("hej apa", "hej bepa", "kungen", "Erik"), no_match = FALSE)
```

change_col_name *Change one column name of a data.frame*

Description

A data.frame can, by definition, have non unique column names. To change a column name to a new name that is already present in the data.frame might therefore lead to undesiered results. This function handles this problem by dropping the original column cousing the name clash.

Usage

```
change_col_name(x, old_name, new_name, warning = FALSE)
```

Arguments

x	data.frame with a column name to change
old_name	name (as character string) of column in x that should be changed.
new_name	new name (as character string) to use instead of old_name
warning	should a warning be given if a name clash occours (FALSE by default)

Value

The same data.frame but with one column named changed. (Note that the output might have one column less if dropped after name clash.)

Examples

```
ab <- ba <- data.frame(a = letters[1:10], b = 1:10)

# One "traditional" way to change a column name
names(ab)[names(ab) == "a"] <- "b"
names(ab)
ab$b # Returns the first column with name "b"

# Using change_col_names instead:
change_col_name(ba, "a", "b")
```

clean_text *Clean/standardize text*

Description

Removes punctuation and spaces from character string. Also makes it lower case.

Usage

```
clean_text(string)
```

Arguments

string	a character string to "clean"

Value

the cleaned character string (no punctuation, spaces or capital letters)

See Also

best_match

Examples

```
clean_text("HELLO_World!!!")
```

create_s3_method *Template functions to generate basic S3 methods for new classes*

Description

`create_s3_method` creates a method that applies `NextMethod` but that also keeps additional attributes (such as class). `create_s3_print` creates a print method.

Usage

```
create_s3_method(generic = NULL, object = NULL)

create_s3_print(fun, ...)
```

Arguments

generic, object	as described for `NextMethod`
fun	Function to transform object before print (probably `as.character`, `as.numeric` or similair).
...	additional arguments passed to print method

Details

Don't forget to also create for example a data.frame method by

`as.data.frame.xxx <- as.data.frame.vector`

Value

S3-method.

Examples

```
a <- structure(1:10, class = c("b", "numeric"))
a[3] # Normal subsetting makes a loose its attributes
`[.b` <- create_s3_method("[")
print.b <- create_s3_print(as.numeric)
a[3] # attributes preserved even if we can't see them
str(a[3])
```

---

cut.integer *Convert integer vector to Factor*

Description

S3-method for cut applied to integer vectors where all outcome factors are integer intervals.

Usage

```
## S3 method for class 'integer'
cut(x, ...)
```

Arguments

x	integer vector
...	further arguments passed to or from other methods

Value

If `cut.default(x, ...)` returns only intger intervals, these are formated in a more natural way and returned as an ordered factor. If non integer interval limits occur, the output of `cut.default(x, ...)` is retured as is.

Examples

```
cut.default(1:100, seq(0, 100, 20)) # Gives a quite unnatural output
cut(1:100, seq(0, 100, 20)) # Gives nicer and ordered output
cut(1:10, 3) # no integer intervals and therefor same as cut.default
```

exceed_threshold *Check if transformation/coercing of a vector is good enaough*

Description

This function is primarly aimed to check if the transformation of a vector was succesfull enough to return the transformed value instead of the original.

Usage

```
exceed_threshold(original, transformed, threshold = 0.9, force = FALSE,
  ask = FALSE, var_name = "the input vector")
```

Arguments

original	the original vector
transformed	the transformed vector with NA-values for non transformed values
threshold	is a numeric value in [0,1] specifying the proportion of cells in transformed that should be recognised as correctly coerced to accept the new class. This does not effect the function output (except when force = TRUE) but will have some diagnostic benefits.
force	Should a candidate vector (candidate according to threshold) be forced to its suggested class (with non-coercable elements set to NA). FALSE by default but if the function is called interactivelly, the user will also have the option to set force = TRUE on the fly.
ask	this argument gives you the chance to interactively inspect your data and specify if a column is a date or not, on the fly. This is FALSE by default for as.Dates.default but TRUE for as.Dates.dataframe. It only applies when the function is runed interactively and only when force == FALSE.
var_name	a name for the object to be used in messages (you could probably just leave this as default, NULL; it is mostly used for internal purposes!).

Value

Either original or transformed.

Examples

```
x <- c(rep("2012-01-01", 9), "foo")
exceed_threshold(x, as.Date(x))
exceed_threshold(x, as.Date(x), force = TRUE)
exceed_threshold(x, as.Date(x), ask = TRUE)
exceed_threshold(x, as.Date(x), threshold = 1)
exceed_threshold(x, as.Date(x), var_name = "bar", force = TRUE)

x <- c(1:9, "baz")
exceed_threshold(x, suppressWarnings(as.numeric(x)))
```

`findvar` *Find variables by name*

Description

Function to seacrh for a variable by its name. See the "Value" section for more details on the different functions.

Usage

```
findvar_fun(df, ...)

findvar_in_df(pattern, df, ...)

findvar_anywhere(pattern, envir = .GlobalEnv, ...)
```

Arguments

`df`	A data.frame from where the column names should be identified when returned function applied
`...`	Arguments passed to `grep`
`pattern`	A character string with name (or part of name) of the variables to find.
`envir`	environment holding data.frames where to search for the variables (the Global environment as default).

Value

- `findvar_fun`: A function with argument `param` to search for `param` in `df`. See example!
- `findvar_in_df`: A vector with variable names from df matching the pattern.
- `findvar_anywhere`: Does not return anything but prints a message where variables matching the pattern can be found.

Examples

```
find_cars <- findvar_fun(cars)
find_cars("sp")

findvar_in_df("sp", cars)

cars <- cars; iris <- iris
findvar_anywhere("petal")
```

is.inca *Are we running on INCA?*

Description

Are we running on INCA?

Usage

```
is.inca()
```

Value

TRUE if we are running on INCA, FALSE otherwise

Examples

```
is.inca()
```

is.scalar_in *Test if scalar is in intervall*

Description

Test if scalar is in intervall

Usage

```
is.scalar_in(left, right)

is.scalar_in01(x)
```

Arguments

left, right	arguments passed to between
x	R object to be tested, most likely a numeric vector of length one (other formats are allowed but will always return FALSE).

Value

is.scalar_in01 returns TRUE if x is an atomic vector of length one and 0 <= as_numeric(x) <= 1. is.scalar_in return a function similair to is.scalar_in01 but with specified boundaries.

Examples

```
is.scalar_in01(.5) # TRUE
is.scalar_in01(5) # FALSE
is.scalar_in01(seq(0,1,.1)) # FALSE

is_scalar_in09 <- is.scalar_in(0,9)
is_scalar_in09(5) # TRUE
```

`is.wholenumber` *Test if a numeric vector consists of whole numbers*

Description

Function borrowed from the examle section for `integer`.

Usage

```
is.wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

`x`	a numeric vector
`tol`	How much is `x` allowed to deviate from `round(x)` to be a whole number.

Value

Logical vector with same length as `x`.

Examples

```
is.wholenumber(1) # is TRUE
(x <- seq(1, 5, by = 0.5) )
is.wholenumber( x ) #-->  TRUE FALSE TRUE ...
```

`lownames` *Make all names in data.frame lower case*

Description

Tests are also performed so that all column names will stay unique!

Usage

```
lownames(df)
```

Arguments

df	A data.frame, possibly with some names with capital letters

Value

df is returned unchanged, except that capital letters in names are changed to lower case.

Examples

```
df <- data.frame(Hello = 1:10, World = 1:10)
lownames(df)
```

---

make_r_script *Dump skript together with all functions from package*

Description

Dump skript together with all functions from package

Usage

```
make_r_script(script = NULL, package, outfile = "./inca_r_script.R",
  all = TRUE)
```

Arguments

script	connection with script (file) to append tp function defenitions
package	name of package (as character)
outfile	filename for dump file
all	should all (even non exported) objects from the package be exported?

Value

nothing (function called for its side effects)

psum *Parallel sum*

Description

This functin is to sum, what pmin and pmax is to min and max.

Usage

```
psum(..., na.rm = FALSE)
```

Arguments

...	numeric vectors
na.rm	a logical indicating whether missing values should be removed.

Examples

```
psum(1:10, 1:10, 1:10)
```

rccmisc *Miscellaneous R functions for the Regional Cancer Centers*

Description

The content of the package is quite diversed. It contains various functions used for various purposes.

Details

Use `help(package="rccmisc")` to list all (exported) functions from the package.

Author(s)

Erik Bulow

`safe_ifelse` *A safe alternative to ifelse*

Description

This function is similair to `ifelse` with differences described in the details section.

Usage

```
safe_ifelse(test, yes, no, na_as_false = TRUE, drop.levels = TRUE)
```

Arguments

`test, yes, no`	arguments passed to `ifelse`
`na_as_false`	should `NA` values in `test` be handled as `FALSE`? `TRUE` (which is default) implies that `test & !is.na(test)` must be fullfilled to return values from argument `yes`
`drop.levels`	This only applies when `yes` and `no` are factor variables. The result will then also be a factor. Unused levels (from `yes` and `no` combined) are dropped by default.

Details

`safe_ifelse` differs from `ifelse` in the following ways:

- Both 'yes' and 'no' must be vectors of the same type or class. This ensures that the output will be of correct format.
- Factors can be combined without problem
- The argument `na.rm` makes it easier to handle cases when `cond = NA`

Value

Vector of same length and class as `yes` and `no`.

Examples

```
# Test must be TRUE to return 'yes'
safe_ifelse(NA, 1, 2) ## 2
ifelse(NA, 1, 2) ## NA

# Factors are problematic in ifelse
ifelse(TRUE, as.factor("hello"), 2) ## 1
## Not run:
safe_ifelse(TRUE, as.factor("hello"), 2) ## Error

## End(Not run)
safe_ifelse(TRUE, as.factor("hello"), as.factor(2)) ## hello
safe_ifelse(TRUE, as.factor("hello"), as.factor(2), drop.levels = FALSE)
```

specify_missing *Specify missing values for a vector*

Description

Change specified values to NA

Usage

```
specify_missing(x, ..., default_missing = c("", NA, "blanks"))
```

Arguments

x	vector
...	values that should be changed to NA if found in x
default_missing	a vector with additional default values to change to NA. These are treated the same as ... but are added by default if not removed. A special value "blank" can be used to indicate all empty strings (all characters matching [:blank:], see regex).

Value

x itself but with specified values set to NA.

Examples

```
x <- sample(100)
x[sample(100, 10)] <- 999
specify_missing(x, 999)
```

width *Calculate the width of the range of x*

Description

Calculate the width of the range of x

Usage

```
width(x)
```

Arguments

x	object to calculate range for

Value

The width of the range of x as integer.

Examples

```
width(1:10)
width(c(6748, 234, 2456, 5678))
width(sample(345))
```

Index