

# Package: rapportools (via r-universe)

September 12, 2024

**Maintainer** Gergely Daróczy <daroczig@rapporter.net>

**Title** Miscellaneous (Stats) Helper Functions with Sane Defaults for Reporting

**Type** Package

**Encoding** UTF-8

**Description** Helper functions that act as wrappers to more advanced statistical methods with the advantage of having sane defaults for quick reporting.

**Author** Aleksandar Blagotić <alex@rapporter.net> and Gergely Daróczy <daroczig@rapporter.net>

**Version** 1.1

**Date** 2022-03-22

**BugReports** <https://github.com/rapporter/rapportools/issues>

**License** AGPL-3

**Imports** plyr, pander, reshape2, MASS

**RoxygenNote** 7.1.1

**Repository** <https://rapporter.r-universe.dev>

**RemoteUrl** <https://github.com/rapporter/rapportools>

**RemoteRef** HEAD

**RemoteSha** f45730af9cbdf147cafd7c030602bae2fe915d5

## Contents

adj.rle . . . . .	3
alike.integer . . . . .	3
capitalise . . . . .	4
catn . . . . .	4
fml . . . . .	5
fraction.to.string . . . . .	5
htest . . . . .	6

htest.short . . . . .	7
iqr . . . . .	7
is.boolean . . . . .	8
is.empty . . . . .	8
is.number . . . . .	9
is.string . . . . .	10
is.tabular . . . . .	10
is.variable . . . . .	11
kurtosis . . . . .	12
label . . . . .	12
label<- . . . . .	13
lambda.test . . . . .	14
max . . . . .	15
mean . . . . .	15
median . . . . .	16
messagef . . . . .	16
min . . . . .	17
n . . . . .	17
name . . . . .	18
nmissing . . . . .	18
nvalid . . . . .	19
pct . . . . .	19
percent . . . . .	20
range . . . . .	21
rp.desc . . . . .	21
rp.freq . . . . .	22
rp.outlier . . . . .	23
sd . . . . .	24
se.mean . . . . .	25
skewness . . . . .	25
stopf . . . . .	26
strwidthest . . . . .	26
sum . . . . .	27
synonym . . . . .	27
synonyms . . . . .	28
tocamel . . . . .	28
trim.space . . . . .	29
univar . . . . .	30
var . . . . .	30
vgsub . . . . .	31
warningf . . . . .	31

---

`adj.rle`*Adjacent Values Run Length Encoding*

---

**Description**

Similar to `rle` function, this function detects "runs" of adjacent integers, and displays vector of run lengths and list of corresponding integer sequences.

**Usage**

```
adj.rle(x)
```

**Arguments**

`x` a numeric vector with

**Value**

a list with two elements: vector of run lengths, and another list of values corresponding to generated sequences' lengths.

**Author(s)**

Gabor Grothendieck <ggrothendieck@gmail.com>

**References**

See original thread for more details <https://stackoverflow.com/a/8467446/564164>. Special thanks to Gabor Grothendieck for this one!

---

`alike.integer`*Check integers*

---

**Description**

This function tests if given variable "appears" to be an integer. To qualify as such, two conditions need to be satisfied: it should be stored as `numeric` object, and it should pass regular expression test if it consists only of digits.

**Usage**

```
alike.integer(x)
```

**Arguments**

`x` a numeric variable that is to be tested

**Value**

a logical value that indicates that tested variable "looks like" integer

---

capitalise	<i>Capitalise String</i>
------------	--------------------------

---

**Description**

Capitalises strings in provided character vector

**Usage**

```
capitalise(x)
```

**Arguments**

x                    a character vector to capitalise

**Value**

character vector with capitalised string elements

**Examples**

```
capitalise(c("foo", "bar")) # [1] "Foo" "Bar"
```

---

catn	<i>Concatenate with newline</i>
------	---------------------------------

---

**Description**

A simple wrapper for [cat](#) function that appends newline to output.

**Usage**

```
catn(...)
```

**Arguments**

...                    arguments to be passed to cat function

**Value**

None (invisible NULL).

---

**fml** *Create Formula from Strings*


---

**Description**

Takes multiple character arguments as left and right-hand side arguments of a formula, and concatenates them in a single string.

**Usage**

```
fml(left, right, join.left = " + ", join.right = " + ")
```

**Arguments**

left	a string with left-hand side formula argument
right	a character vector with right-hand side formula arguments
join.left	concatenation string for elements of character vector specified in left
join.right	concatenation string for elements of character vector specified in right

**Examples**

```
fml("hp", c("am", "cyl")) # "hp ~ am + cyl"
```

---

**fraction.to.string** *A fraction in ordinary English language*


---

**Description**

A fraction in ordinary English language

**Usage**

```
fraction.to.string(x)
```

**Arguments**

x	numeric
---	---------

**Value**

string

---

htest	<i>Hypothesis Tests</i>
-------	-------------------------

---

### Description

This function uses `htest.short`, to extract statistic and p-value from htest-classed object. Main advantage of using htest is that it's vectorised, and can accept multiple methods.

### Usage

```
htest(
  x,
  ...,
  use.labels = getOption("rapport.use.labels"),
  use.method.names = TRUE,
  colnames = c("Method", "Statistic", "p-value")
)
```

### Arguments

x	arguments to be passed to function specified in test
...	additional arguments for function specified in test
use.labels	a logical value indicating whether variable labels should be placed in row names. If set to FALSE, output of <code>deparse(substitute(x))</code> will be used.
use.method.names	use the string provided in method attribute of htest object
colnames	a character string containing column names

### Details

Default parameters are read from options:

- 'rapport.use.labels'.

### Value

a data.frame with applied tests in rows, and their results (statistic and p-value) in columns

### Examples

```
## Not run:
library(nortest)
htest(rnorm(100), shapiro.test)
htest(rnorm(100), lillie.test, ad.test, shapiro.test)
htest(mtcars, lillie.test)
htest(mtcars, lillie.test, ad.test, shapiro.test)

## End(Not run)
```

---

hstest.short	<i>Extract Values from hstest Objects</i>
--------------	---

---

**Description**

Extract value of statistic and its p-value from hstest object.

**Usage**

```
hstest.short(x)
```

**Arguments**

x                    hstest-class object

**Value**

named numeric vector with the value of statistic and its p-value

**Examples**

```
## Not run:  
hstest.short(shapiro.test(rnorm(100)))  
  
## End(Not run)
```

---

iqr	<i>Interquartile Range</i>
-----	----------------------------

---

**Description**

Calculates interquartile range of given variable. See [univar](#) for details.

**Usage**

```
iqr(...)
```

**Arguments**

...                    parameters to be passed to univar function

**Value**

a numeric value with variable's interquartile range

---

is.boolean	<i>Boolean</i>
------------	----------------

---

**Description**

Checks if provided object is a boolean i.e. a length-one logical vector.

**Usage**

```
is.boolean(x)
```

**Arguments**

x	an object to check
---	--------------------

**Value**

a logical value indicating whether provided object is a boolean

**Examples**

```
## Not run:
  is.boolean(TRUE)           # [1] TRUE
  # the following will work on most systems, unless you have tweaked global Rprofile
  is.boolean(T)             # [1] TRUE
  is.boolean(1)             # [1] FALSE
  is.string(c("foo", "bar")) # [1] FALSE

## End(Not run)
```

---

is.empty	<i>Empty Value</i>
----------	--------------------

---

**Description**

Rails-inspired helper that checks if vector values are "empty", i.e. if it's: NULL, zero-length, NA, NaN, FALSE, an empty string or  $\emptyset$ . Note that unlike its native R `is.<something>` sibling functions, `is.empty` is vectorised (hence the "values").

**Usage**

```
is.empty(x, trim = TRUE, ...)
```

**Arguments**

x	an object to check its emptiness
trim	trim whitespace? (TRUE by default)
...	additional arguments for <a href="#">sapply</a>



**Examples**

```
## Not run:
is.empty(NULL)      # [1] TRUE
is.empty(c())       # [1] TRUE
is.empty(NA)        # [1] TRUE
is.empty(NaN)       # [1] TRUE
is.empty("")        # [1] TRUE
is.empty(0)         # [1] TRUE
is.empty(0.00)      # [1] TRUE
is.empty(" ")       # [1] TRUE
is.empty("foobar")  # [1] FALSE
is.empty(" ", trim = FALSE) # [1] FALSE
# is.empty is vectorised!
all(is.empty(rep("", 10))) # [1] TRUE
all(is.empty(matrix(NA, 10, 10))) # [1] TRUE

## End(Not run)
```

---

is.number	<i>Numbers</i>
-----------	----------------

---

**Description**

Checks if provided object is a number, i.e. a length-one numeric vector.

**Usage**

```
is.number(x, integer = FALSE)
```

**Arguments**

x	an object to check
integer	logical: check if number is integer

**Value**

a logical value indicating whether provided object is a string

**Examples**

```
is.number(3)      # [1] TRUE
is.number(3:4)    # [1] FALSE
is.number("3")    # [1] FALSE
is.number(NaN)    # [1] TRUE
is.number(NA_integer_) # [1] TRUE
```

---

`is.string`*Strings*

---

**Description**

Checks if provided object is a string i.e. a length-one character vector.

**Usage**

```
is.string(x)
```

**Arguments**

x                    an object to check

**Value**

a logical value indicating whether provided object is a string

**Examples**

```
is.string("foobar")            # [1] TRUE
is.string(1)                    # [1] FALSE
is.string(c("foo", "bar"))    # [1] FALSE
```

---

`is.tabular`*Tabular Structure*

---

**Description**

Checks if object has "tabular" structure (not to confuse with [table](#)) - in this particular case, that means [matrix](#) and [data.frame](#) objects only.

**Usage**

```
is.tabular(x)
```

**Arguments**

x                    an object to be checked for "tabular" format

**Value**

a logical value indicating that provided object has tabular structure

**Examples**

```

is.tabular(HairEyeColor[, , 1]) # [1] TRUE
is.tabular(mtcars)              # [1] TRUE
is.tabular(table(mtcars$cyl))  # [1] FALSE
is.tabular(rnorm(100))         # [1] FALSE
is.tabular(LETTERS)            # [1] FALSE
is.tabular(pi)                 # [1] FALSE

```

---

is.variable	<i>Variables</i>
-------------	------------------

---

**Description**

From *rapport*'s point of view, a *variable* is a non-NULL atomic vector that has no dimension attribute (see `dim` for details). This approach bypasses factor issues with `is.vector`, and also eliminates multidimensional vectors, such as matrices and arrays.

**Usage**

```
is.variable(x)
```

**Arguments**

`x` an object to be checked for "variable" format

**Value**

a logical value indicating that provided object is a "variable"

**Examples**

```

is.variable(rnorm(100)) # [1] TRUE
is.variable(LETTERS)   # [1] TRUE
is.variable(NULL)      # [1] FALSE
is.variable(mtcars)    # [1] FALSE
is.variable(HairEyeColor[, , 1]) # [1] FALSE
is.variable(list())    # [1] FALSE

```

---

kurtosis	<i>Kurtosis</i>
----------	-----------------

---

**Description**

Calculates kurtosis coefficient for given variable (see `is.variable`), matrix or a `data.frame`.

**Usage**

```
kurtosis(x, na.rm = TRUE)
```

**Arguments**

x	a variable, matrix or a <code>data.frame</code>
na.rm	should NAs be removed before computation?

**References**

Tenjovic, L. (2000). Statistika u psihologiji - prirucnik. Centar za primenjenu psihologiju.

**Examples**

```
set.seed(0)
x <- rnorm(100)
kurtosis(x)
kurtosis(matrix(x, 10))
kurtosis(mtcars)
rm(x)
```

---

label	<i>Get Variable Label</i>
-------	---------------------------

---

**Description**

This function returns character value previously stored in variable's `label` attribute. If none found, and `fallback` argument is set to `TRUE` (default), the function returns object's name (retrieved by `deparse(substitute(x))`), otherwise `NA` is returned with a warning notice.

**Usage**

```
label(x, fallback = TRUE, simplify = TRUE)
```

**Arguments**

x	an R object to extract labels from
fallback	a logical value indicating if labels should fallback to object name(s)
simplify	coerce results to a vector ( <code>TRUE</code> by default), otherwise, a list is returned

**Value**

a character vector with variable's label(s)

**Examples**

```
## Not run:
x <- rnorm(100)
label(x)      # returns "x"
label(x, FALSE) # returns NA and issues a warning

label(mtcars$hp) <- "Horsepower"
label(mtcars)      # returns "Horsepower" instead of "hp"
label(mtcars, FALSE) # returns NA where no labels are found
label(sleep, FALSE) # returns NA for each variable and issues a warning

## End(Not run)
```

---

label<-	<i>Set Variable Label</i>
---------	---------------------------

---

**Description**

This function sets a label to a variable, by storing a character string to its label attribute.

**Usage**

```
label(var) <- value
```

**Arguments**

var	a variable (see <a href="#">is.variable</a> for details)
value	a character value that is to be set as variable label

**See Also**

[label](#)

**Examples**

```
## Not run:
label(mtcars$mpg) <- "fuel consumption"
x <- rnorm(100)
(label(x) <- "pseudo-random normal variable")

## End(Not run)
```

---

lambda.test	<i>Goodman and Kruskal's lambda</i>
-------------	-------------------------------------

---

**Description**

Computes Goodman and Kruskal's lambda for given table.

**Usage**

```
lambda.test(table, direction = 0)
```

**Arguments**

table	a table of two variables or a data.frame representation of the cross-table of the two variables without marginals
direction	numeric value of c(0, 1, 2) where 1 means the lambda value computed for row, 2 for columns and 0 for both

**Value**

computed lambda value(s) for row/col of given table

**References**

- Goodman, L.A., Kruskal, W.H. (1954) Measures of association for cross classifications. Part I. *Journal of the American Statistical Association* **49**, 732–764

**Examples**

```
## Not run:
## quick example
x <- data.frame(x = c(5, 4, 3), y = c(9, 8, 7), z = c(7, 11, 22), zz = c(1, 15, 8))
lambda.test(x) # 0.1 and 0.18333
lambda.test(t(x)) # 0.18333 and 0.1

## historical data (see the references above: p. 744)
men.hair.color <- data.frame(
  b1 = c(1768, 946, 115),
  b2 = c(807, 1387, 438),
  b3 = c(189, 746, 288),
  b4 = c(47, 53, 16)
)
row.names(men.hair.color) <- paste0('a', 1:3)
lambda.test(men.hair.color)
lambda.test(t(men.hair.color))

## some examples on mtcars
lambda.test(table(mtcars$am, mtcars$gear))
lambda.test(table(mtcars$gear, mtcars$am))
```

```
lambda.test(table(mtcars$am, mtcars$gear), 1)
lambda.test(table(mtcars$am, mtcars$gear), 2)

## End(Not run)
```

---

max

*Maximum*


---

### Description

Returns the maximum of all values in a vector by passing `max` as fn argument to `univar` function.

### Usage

```
max(...)
```

### Arguments

... parameters to be passed to univar function

### Value

a numeric value with maximum value

---

mean

*Mean*


---

### Description

Calculates mean of given variable by passing `sum` as fn argument to `univar` function.

### Usage

```
mean(...)
```

### Arguments

... parameters to be passed to univar function

### Value

a numeric value with variable's mean

median

*Median*

---

**Description**

Calculates median of given variable. See [univar](#) for details.

**Usage**

```
median(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with variable's median

---

messagef

*Send Message with String Interpolated Messages*

---

**Description**

Combines warning with `sprintf` thus allowing string interpolated diagnostic messages.

**Usage**

```
messagef(s, ...)
```

**Arguments**

s a character vector of format strings

... values to be interpolated

**Examples**

```
## Not run:  
messagef("%.3f is not larger than %d and/or smaller than %d", pi, 10, 40)  
  
## End(Not run)
```



---

min	<i>Minimum</i>
-----	----------------

---

**Description**

Returns the minimum of all values in a vector by passing `min` as fn argument to `univar` function.

**Usage**

```
min(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with minimum value

---

n	<i>Number of Cases</i>
---	------------------------

---

**Description**

Returns the number of valid (non-NA) values in a variable. This is a wrapper around `univar` function with `length` function passed in fn argument, but with missing values previously removed. However, it's not possible to cancel NA omission with this function (doing so will yield error).

**Usage**

```
n(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with number of valid (non-NA) vector elements

name *Variable Name*

---

**Description**

This function returns character value previously stored in variable's name attribute. If none found, the function defaults to object's name.

**Usage**

```
name(x)
```

**Arguments**

x an R (atomic or data.frame/list) object to extract names from

**Value**

a character value with variable's label

**Examples**

```
## Not run:  
name(mtcars$am)  
x <- 1:10  
name(x)  
  
## End(Not run)
```

---

nmissing *Number of Missing Cases*

---

**Description**

Returns a number of missing (NA) values in a variable. This is a wrapper around [univar](#) function with anonymous function passed to count number of NA elements in a variable.

**Usage**

```
nmissing(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with number of missing vector elements

---

nvalid	<i>Number of Valid Cases</i>
--------	------------------------------

---

**Description**

Returns the number of valid (non-NA) values in a variable. This is a wrapper around `univar` function with `length` function passed in `fn` argument, but with missing values previously removed. However, it's not possible to cancel NA omission with this function (doing so will yield error).

**Usage**

```
nvalid(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with number of valid (non-NA) vector elements

---

pct	<i>Percent</i>
-----	----------------

---

**Description**

Appends a percent sign to provided numerical value. Rounding is carried out according to value passed in `decimals` formal argument (defaults to value specified in `panderOptions('digits')`).

**Usage**

```
pct(
  x,
  digits = panderOptions("digits"),
  type = c("percent", "%", "proportion"),
  check.value = TRUE
)
```

**Arguments**

x	a numeric value that is to be rendered to percent
digits	an integer value indicating number of decimal places
type	a character value indicating whether percent or proportion value was provided (partial match is allowed)
check.value	perform a sanity check to see if provided numeric value is correct (defaults to TRUE)

**Value**

a character value with formatted percent

---

percent	<i>Percent</i>
---------	----------------

---

**Description**

Calculates percentage of cases for provided variable and criteria specified in subset argument. Function accepts numeric, factor and logical variables for x parameter. If numeric and/or factor is provided, subsetting can be achieved via subset argument. Depending on value of na.rm argument, either valid (na.rm = TRUE) or all cases (na.rm = FALSE) are taken into account. By passing logical variable to x, a sum of (TRUE) elements is calculated instead, and valid percents are used (NA are excluded).

**Usage**

```
percent(x, subset = NULL, na.rm = TRUE, pct = FALSE, ...)
```

**Arguments**

x	a numeric variable to be summarised
subset	an expression that evaluates to logical vector (defaults to NULL)
na.rm	should missing values be
pct	print percent string too?
...	additional arguments for <a href="#">pct</a> function

**Value**

a numeric or string depending on the value of pct

**Examples**

```
## Not run:  
set.seed(0)  
x <- sample(5, 100, replace = TRUE)  
percent(x > 2)  
  
## End(Not run)
```

---

range	<i>Range</i>
-------	--------------

---

**Description**

Calculates difference between the largest and the smallest value in a vector. See [univar](#) for details.

**Usage**

```
range(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with calculated range

---

rp.desc	<i>Descriptive Statistics</i>
---------	-------------------------------

---

**Description**

Aggregate table of descriptives according to functions provided in `fn` argument. This function follows `melt/cast` approach used in `reshape` package. Variable names specified in `measure.vars` argument are treated as `measure.vars`, while the ones in `id.vars` are treated as `id.vars` (see [melt.data.frame](#) for details). Other its formal arguments match with corresponding arguments for `cast` function. Some post-processing is done after reshaping, in order to get pretty row and column labels.

**Usage**

```
rp.desc(
  measure.vars,
  id.vars = NULL,
  fn,
  data = NULL,
  na.rm = FALSE,
  margins = TRUE,
  total.name = "Total",
  use.labels = getOption("rapport.use.labels")
)
```

**Arguments**

<code>measure.vars</code>	either a character vector with variable names from data, a numeric vector, or a <code>data.frame</code>
<code>id.vars</code>	same rules apply as in <code>measure.vars</code> , but defaults to <code>NULL</code>
<code>fn</code>	a list with functions or a character vector with function names
<code>data</code>	a <code>data.frame</code> holding variables specified in <code>id.vars</code> and <code>measure.vars</code>
<code>na.rm</code>	a logical value indicating whether NA values should be removed
<code>margins</code>	should margins be included?
<code>total.name</code>	a character string with name for "grand" margin (defaults to "Total")
<code>use.labels</code>	use labels instead of variable names in table header (handle with care, especially if you have lengthy labels). Defaults to value specified in <code>rapport.use.labels</code> option.

**Value**

a `data.frame` with aggregated data

**Examples**

```
rp.desc("cyl", NULL, c(mean, sd), mtcars)
rp.desc("cyl", "am", c(mean, sd), mtcars, margins = TRUE)
rp.desc("hp", c("am", "gear"), c("Average" = mean, "Deviation" = sd), mtcars)
```

---

rp.freq

*Frequency Table*

---

**Description**

Display frequency table with counts, percentage, and cumulatives.

**Usage**

```
rp.freq(
  f.vars,
  data,
  na.rm = TRUE,
  include.na = FALSE,
  drop.unused.levels = FALSE,
  count = TRUE,
  pct = TRUE,
  cumul.count = TRUE,
  cumul.pct = TRUE,
  total.name = "Total",
  reorder = FALSE
)
```

**Arguments**

f.vars	a character vector with variable names
data	a data.frame
na.rm	should missing values be removed?
include.na	should missing values be included in frequency table?
drop.unused.levels	should empty level combinations be left out
count	show frequencies?
pct	show percentage?
cumul.count	show cumulative frequencies?
cumul.pct	show cumulative percentage?
total.name	a sting containing footer label (defaults to "Total")
reorder	reorder the table based on frequencies?

**Value**

a data.frame with a frequency table

**Examples**

```
## Not run:
rp.freq(c("am", "cyl", "vs"), mtcars)

## End(Not run)
```

---

rp.outlier

*Outlier test*

---

**Description**

A simple test for outliers. This functions returns all extreme values (if any) found in the specified vector.

**Usage**

```
rp.outlier(x)
```

**Arguments**

x	a numeric vector of values
---	----------------------------

**Value**

vector of outlier values

## References

Credit goes to PaulHurleyuk: <https://stackoverflow.com/a/1444548/564164>

- Lund, R. E. 1975, "Tables for An Approximate Test for Outliers in Linear Models", Technometrics, vol. 17, no. 4, pp. 473-476.
- Prescott, P. 1975, "An Approximate Test for Outliers in Linear Models", Technometrics, vol. 17, no. 1, pp. 129-132.

## Examples

```
## Not run:  
rp.outlier(mtcars$hp)  
rp.outlier(c(rep(1,100), 200))  
rp.outlier(c(rep(1,100), 200,201))  
  
## End(Not run)
```

---

sd	<i>Standard Deviation</i>
----	---------------------------

---

## Description

Calculates standard deviation of given variable. See [univar](#) for details.

## Usage

```
sd(...)
```

## Arguments

```
...           parameters to be passed to univar function
```

## Value

a numeric value with variable's standard deviation



---

se.mean	<i>Standard Error of Mean</i>
---------	-------------------------------

---

**Description**

Calculates standard error of mean for given variable. See [univar](#) for details.

**Usage**

```
se.mean(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with standard error of mean

---

skewness	<i>Skewness</i>
----------	-----------------

---

**Description**

Calculates skewness coefficient for given variable (see [is.variable](#)), matrix or a data.frame.

**Usage**

```
skewness(x, na.rm = TRUE)
```

**Arguments**

x a variable, matrix or a data.frame  
na.rm should NAs be removed before computation?

**References**

Tenjovic, L. (2000). Statistika u psihologiji - prirucnik. Centar za primenjenu psihologiju.

**Examples**

```
set.seed(0)
x <- rnorm(100)
skewness(x)
skewness(matrix(x, 10))
skewness(mtcars)
rm(x)
```

---

stopf *Stop Execution with String Interpolated Messages*

---

### Description

This helper combines stop function with sprintf thus allowing string interpolated messages when execution is halted.

### Usage

```
stopf(s, ...)
```

### Arguments

s                    a character vector of format strings  
 ...                  values to be interpolated

### Value

a string containing message that follows execution termination

### Examples

```
## Not run:
stopf("%.3f is not larger than %d and/or smaller than %d", pi, 10, 40)

## End(Not run)
```

---

strwidthest *Estimate string width*

---

### Description

Pre-computed width of strings without actually calling to **graphics** and new.plot. The function can only handle base ASCII characters and default width of those is estimated by using standard 12 pt serif on a standard plot. Non-ASCII characters are replaced by an underscore.

### Usage

```
strwidthest(s)
```

### Arguments

s                    string

**Value**

numeric value representing the total width of the provided string in millimeters

**Examples**

```
## Not run:
strwidththrel('R') # 71
strwidththrel('R is awesome!') # 635

## End(Not run)
```

---

sum

*Sum*


---

**Description**

Returns the sum of variable's elements, by passing `sum` as fn argument to `univar` function.

**Usage**

```
sum(...)
```

**Arguments**

... parameters to be passed to univar function

**Value**

a numeric value with sum of vector elements

---

synonym

*Synonym*


---

**Description**

Synonym

**Usage**

```
synonym(word)
```

**Arguments**

word a word to look-up in 'rapportools::synonyms'

**Value**

a synonym if found in 'rapportools::synonyms' words

**Examples**

```
synonym('package')
synonym('bar')
```

---

synonyms

*Get or set synonyms list*

---

**Description**

Without the `l` parameter, this function returns the saved list of synonym words. If `l` is set, then this word list is saved for future use.

**Usage**

```
synonyms(l)
```

**Arguments**

`l` a grouped list of words

**Examples**

```
{
  synonyms(list(c('package', 'library'), c('foo', 'bar', 'baz')))
  synonyms()
}
```

---

tocamel

*CamelCase*

---

**Description**

Convert character vector to camelcase - capitalise first letter of each word.

**Usage**

```
tocamel(x, delim = "[^[:alnum:]]", upper = FALSE, sep = "", ...)
```

**Arguments**

x	a character vector to be converted to camelcase
delim	a string containing regular expression word delimiter
upper	a logical value indicating if the first letter of the first word should be capitalised (defaults to FALSE)
sep	a string to separate words
...	additional arguments to be passed to <code>strsplit</code>

**Value**

a character vector with strings put in camelcase

**Examples**

```

tocamel("foo.bar")
## [1] "fooBar"

tocamel("foo.bar", upper = TRUE)
## [1] "FooBar"

tocamel(c("foobar", "foo.bar", "camel_case", "a.b.c.d"))
## [1] "foobar" "fooBar" "camelCase" "aBCD"

```

---

trim.space

*Trim Spaces*


---

**Description**

Removes leading and/or trailing space(s) from a character vector. By default, it removes both leading and trailing spaces.

**Usage**

```

trim.space(
  x,
  what = c("both", "leading", "trailing", "none"),
  space.regex = "[:space:]",
  ...
)

```

**Arguments**

x	a character vector which values need whitespace trimming
what	which part of the string should be trimmed. Defaults to both which removes trailing and leading spaces. If none, no trimming will be performed.
space.regex	a character value containing a regex that defines a space character
...	additional arguments for <code>gsub</code> function

**Value**

a character vector with (hopefully) trimmed spaces

---

univar

*Descriptive Statistics*

---

**Description**

This function operates only on vectors or their subsets, by calculating a descriptive statistic specified in `fn` argument.

**Usage**

```
univar(x, subset = NULL, fn, na.rm = TRUE, ...)
```

**Arguments**

<code>x</code>	a numeric variable to be summarised
<code>subset</code>	an expression that evaluates to logical vector (defaults to <code>NULL</code> , in which case the function specified in <code>fn</code> is applied on a vector)
<code>fn</code>	a function or a function name to be applied on a variable or it's subset
<code>na.rm</code>	a logical value indicating whether NA's should be removed (defaults to <code>TRUE</code> )
<code>...</code>	additional arguments for function specified in <code>fn</code>

**Value**

a numeric

---

var

*Variance*

---

**Description**

Calculates variance of given variable. See [univar](#) for details.

**Usage**

```
var(...)
```

**Arguments**

<code>...</code>	parameters to be passed to <code>univar</code> function
------------------	---

**Value**

a numeric value with variable's variance

---

vgsub	<i>Vectorised String Replacement</i>
-------	--------------------------------------

---

**Description**

A simple wrapper for `gsub` that replaces all patterns from `pattern` argument with ones in `replacement` over vector provided in argument `x`.

**Usage**

```
vgsub(pattern, replacement, x, ...)
```

**Arguments**

<code>pattern</code>	see eponymous argument for <code>gsub</code> function
<code>replacement</code>	see eponymous argument for <code>gsub</code> function
<code>x</code>	see eponymous argument for <code>gsub</code> function
<code>...</code>	additional arguments for <code>gsub</code> function

**Value**

a character vector with string replacements

**References**

See original thread for more details <https://stackoverflow.com/a/6954308/564164>. Special thanks to user Jean-Robert for this one!

---

warningf	<i>Send Warning with String Interpolated Messages</i>
----------	---

---

**Description**

Combines `warning` with `sprintf` thus allowing string interpolated warnings.

**Usage**

```
warningf(s, ...)
```

**Arguments**

<code>s</code>	a character vector of format strings
<code>...</code>	values to be interpolated

**Examples**

```
## Not run:  
warningf("%.3f is not larger than %d and/or smaller than %d", pi, 10, 40)  
  
## End(Not run)
```



# Index

adj.rle, [3](#)  
alike.integer, [3](#)

capitalise, [4](#)  
cast, [21](#)  
cat, [4](#)  
catn, [4](#)

data.frame, [10](#)

fml, [5](#)  
fraction.to.string, [5](#)

gsub, [29](#), [31](#)

htest, [6](#)  
htest.short, [6](#), [7](#)

IQR (iqr), [7](#)  
iqr, [7](#)  
is.boolean, [8](#)  
is.empty, [8](#)  
is.number, [9](#)  
is.string, [10](#)  
is.tabular, [10](#)  
is.variable, [11](#), [12](#), [13](#), [25](#)  
is.vector, [11](#)

kurtosis, [12](#)

label, [12](#), [13](#)  
label<-, [13](#)  
lambda.test, [14](#)  
length, [17](#), [19](#)

matrix, [10](#)  
max, [15](#), [15](#)  
mean, [15](#)  
median, [16](#)  
melt.data.frame, [21](#)  
messagef, [16](#)

min, [17](#), [17](#)

n, [17](#)  
name, [18](#)  
nmissing, [18](#)  
numeric, [3](#)  
nvalid, [19](#)

pct, [19](#), [20](#)  
percent, [20](#)

range, [21](#)  
rle, [3](#)  
rp.desc, [21](#)  
rp.freq, [22](#)  
rp.iqr (iqr), [7](#)  
rp.label (label), [12](#)  
rp.label<- (label<-), [13](#)  
rp.max (max), [15](#)  
rp.mean (mean), [15](#)  
rp.median (median), [16](#)  
rp.min (min), [17](#)  
rp.missing (nmissing), [18](#)  
rp.n (n), [17](#)  
rp.name (name), [18](#)  
rp.outlier, [23](#)  
rp.percent (percent), [20](#)  
rp.range (range), [21](#)  
rp.sd (sd), [24](#)  
rp.se.mean (se.mean), [25](#)  
rp.sum (sum), [27](#)  
rp.valid (nvalid), [19](#)  
rp.var (var), [30](#)

sapply, [8](#)  
sd, [24](#)  
se.mean, [25](#)  
skewness, [25](#)  
stopf, [26](#)  
strwidthest, [26](#)

sum, [15](#), [27](#), [27](#)  
synonym, [27](#)  
synonyms, [28](#)

table, [10](#)  
tocamel, [28](#)  
trim.space, [29](#)

univar, [7](#), [15–19](#), [21](#), [24](#), [25](#), [27](#), [30](#), [30](#)

var, [30](#)  
vgsb, [31](#)

warningf, [31](#)