

# Package ‘skMisc’

January 4, 2021

**Title** Sercan Kahveci's Miscellaneous Functions

**Version** 0.01

**Description** Contains a wide range of functions.

**Depends** R (≥ 3.6.1), magrittr, dplyr, doParallel, lmerTest

**Imports** tidyr, knitr, quanteda

**License** GPL-3

**BugReports** <https://github.com/Spiritspeak/skMisc/issues>

**LazyData** true

**ByteCompile** true

**RoxygenNote** 7.1.1

## R topics documented:

AnovaTable . . . . .	2
clamp . . . . .	3
coerce . . . . .	3
colVars . . . . .	4
combobulate . . . . .	4
compcorr . . . . .	5
CorrCrunch . . . . .	5
CorTable . . . . .	6
df.init . . . . .	7
ExpandFormula . . . . .	7
ExtractRandomTerms . . . . .	8
FindTopTerms . . . . .	8
LevenshteinDistance . . . . .	9
logit.weightfun . . . . .	9
OLcrunch . . . . .	10
pair . . . . .	11
read.csv.folder . . . . .	11
RemoveTopTerms . . . . .	12
retype . . . . .	12
smoothvect . . . . .	13
splitColumn . . . . .	14
theme_pecher . . . . .	14
tokens_compound_stepwise . . . . .	15

TransformPlots . . . . .	15
trypackages . . . . .	16
verify_types . . . . .	16
wtd.median . . . . .	17
<b>Index</b>	<b>18</b>

---

AnovaTable	<i>Compare multilevel models</i>
------------	----------------------------------

---

## Description

Compare multilevel models

## Usage

```
AnovaTable(
  ...,
  fullmodel,
  models,
  serial = F,
  suppress = c("AIC", "deviance", "logLik")
)

## S3 method for class 'AnovaTable'
print(x)
```

## Arguments

<code>...</code>	Model objects to be compared
<code>fullmodel</code>	A model to which all other models are to be compared; only use if <code>...</code> is not specified.
<code>models</code>	Models to compare to <code>fullmodel</code> . Only use if <code>...</code> is not specified.
<code>serial</code>	If TRUE, models are compared serially; if false, all models will be compared to the first.
<code>suppress</code>	Character vector of column names to suppress in printed output.

## Value

A data.frame containing model fit metrics such as AIC, BIC, marginal R-squared (the effect size of fixed effects only), conditional R-squared (the effect size of all model terms), loglikelihood, deviance, and a likelihood ratio test.

---

`clamp`*clamp*

---

**Description**

clamp

**Usage**`clamp(val, minval, maxval)`**Arguments**

<code>val</code>	The vector/matrix to clamp
<code>minval</code>	Minimum value; all lower values are clamped to this value
<code>maxval</code>	Maximum value; all higher values are clamped to this value

**Value**

Clamped vector.

**Examples**`clamp(0:10,2,8)`

---

`coerce`*coerce a vector to contain only TRUE and FALSE*

---

**Description**

coerce a vector to contain only TRUE and FALSE

**Usage**`coerce(x, default = FALSE)`**Arguments**

<code>x</code>	Numeric/logical vector/matrix to coerce into TRUE/FALSE
<code>default</code>	default returned value if NULL or NA is encountered

**Value**

logical vector or matrix with only T and F

**Examples**

```

coerce(NULL)
# FALSE

coerce(c(T,F,NA,NA,T))
# T F F F T

coerce(matrix(c(T,T,F,F,NA,NA),nrow=2))
#      [,1] [,2] [,3]
#[1,] TRUE FALSE FALSE
#[2,] TRUE FALSE FALSE

```

---

colVars	<i>Compute column and row variances</i>
---------	---

---

**Description**

Compute column and row variances

**Usage**

```

colVars(x, na.rm = T)

rowVars(x, na.rm = T)

```

**Arguments**

x	an input matrix of data.frame
na.rm	Logical indicating whether NA values should be omitted before variance computation

---

combobulate	<i>Get all possible combinations of strings</i>
-------------	---

---

**Description**

combobulate() returns all possible combinations of the provided character strings, each combination merged into a single string.

**Usage**

```

combobulate(...)

```

**Arguments**

...	Character vectors to combobulate.
-----	-----------------------------------

**Value**

A character vector.

**Examples**

```

combobulate("Hello ",c("Sir","Madam"),",",",c("may I take your order?","what shall it be?"))
# [1] "Hello Sir, may I take your order?"
# [2] "Hello Madam, may I take your order?"
# [3] "Hello Sir, what shall it be?"
# [4] "Hello Madam, what shall it be?"

```

compcorr

*Test if two correlation coefficients significantly differ***Description**

Uses Fisher's r to z transformation, then performs a z-test on the resulting z-scores

**Usage**

```
compcorr(cor1, cor2, n1, n2)
```

**Arguments**

cor1, cor2	Correlation values being compared
n1, n2	Sample sizes of the correlation coefficients

**Value**

List containing the z-score and p-value

**References**

<http://vassarstats.net/rdiff.html>

CorrCrunch

*Analyse the robustness of a correlation***Description**

CorrCrunch() computes the minimum number of cases that need to be removed from a dataset to flip the sign of a correlation coefficient. This can be useful in distinguishing genuine correlations from spurious findings that hinge on one or two outliers. Cases are removed iteratively; in each iteration the case that maximally shrinks the correlation coefficient is removed.

**Usage**

```
CorrCrunch(x, y, verbose = F)
```

**Arguments**

x, y	Numeric vectors to correlate.
verbose	if TRUE, prints verbose output.

**Value**

A list containing the number of cases that need to be removed to flip the sign of the correlation coefficient; the proportion removed cases in the data; and a data.frame without these cases.

**Examples**

```
CorrCrunch(mtcars$mpg,mtcars$wt)
#Holdout needed to flip the sign: 19 (63.33%)
#Final r: 0.01181141
```

---

CorTable	<i>Create a Correlation Table</i>
----------	-----------------------------------

---

**Description**

Create a Correlation Table

**Usage**

```
CorTable(df, rowids, columnids, rowdf, columndf)
```

**Arguments**

**df** A data.frame.

**rowids, columnids** character vectors containing column names from **df** that need to be correlated.

**rowdf, columndf** data.frames whose columns need to be correlated. Either **df**, **rowids**, & **columnids** or **rowdf** & **columndf** are required.

**Value**

A formatted markdown table containing correlation coefficients, p-values, and the number and percentage of cases that need to be removed to flip the sign of each correlation coefficient.

**Examples**

```
CorTable(mtcars,rowids=c("mpg","disp","hp"),columnids=c("drat","wt","qsec"))

CorTable(rowdf=mtcars[,c(1,3,4)],columndf=mtcars[,5:7])
```

---

df.init	<i>Initiate an empty data frame</i>
---------	-------------------------------------

---

**Description**

Initiate an empty data frame

**Usage**

```
df.init(namelist)
```

**Arguments**

namelist      A character vector of column names.

**Value**

A data.frame with 0 rows.

---

ExpandFormula	<i>Parse a lme4 formula and return all main effects and interactions as separate terms</i>
---------------	--

---

**Description**

Parse a lme4 formula and return all main effects and interactions as separate terms

**Usage**

```
ExpandFormula(form)
```

**Arguments**

form

**Value**

The same formula, but with all interactions and main effects as separate terms

**Examples**

```
ExpandFormula(rt ~ pull * target + (pull * target | subjectid))
#rt ~ pull + target + pull:target + (pull + target + pull:target | subjectid)
```

---

ExtractRandomTerms	<i>Extract random terms from a lme4 formula</i>
--------------------	---

---

**Description**

Extract random terms from a lme4 formula

**Usage**

```
ExtractRandomTerms(form)
```

**Arguments**

form	A formula
------	-----------

**Value**

A named list containing character vectors with random terms; names are group variables.

**Examples**

```
ExtractRandomTerms(grade ~ ChildIQ * TeacherSkill * SchoolType +
                    (ChildIQ * TeacherSkill | School))
##$School
#[1] "ChildIQ"          "TeacherSkill"     "ChildIQ:TeacherSkill"
```

---

FindTopTerms	<i>Find all model terms that are not moderated by a higher-order interaction</i>
--------------	--

---

**Description**

Find all model terms that are not moderated by a higher-order interaction

**Usage**

```
FindTopTerms(form)
```

**Arguments**

form	a formula
------	-----------

**Value**

A character vector containing all model terms that are not moderated by a higher-order interaction.

**Examples**

```
FindTopTerms(speed ~ skill + weight * friction)
#[1] "skill"          "weight:friction"
```



---

LevenshteinDistance	<i>Levenshtein distance</i>
---------------------	-----------------------------

---

**Description**

Counts the number of single character deletions, insertions, and substitutions that need to be performed to turn the source string into the target string.

**Usage**

```
LevenshteinDistance(source, target)
```

**Arguments**

source, target    Strings to be compared.

**Value**

The Levenshtein distance between the two strings.

**Examples**

```
LevenshteinDistance("Yoghurt", "Youtube")
```

---

logit.weightfun	<i>Downweight outliers</i>
-----------------	----------------------------

---

**Description**

Computes weights; trials within certain bounds of the mean receive the maximum weight while trials outside these bounds are downweighted to 0 or an optional minimum.

**Usage**

```
logit.weightfun(  
  x,  
  mean = mean(x),  
  s = sd(x),  
  sdist = 3,  
  taper = 10,  
  scale = c("max", "norm"),  
  min = 0  
)
```

**Arguments**

<code>x</code>	A numeric vector
<code>mean</code>	An optional mean of the vector
<code>s</code>	An optional standard deviation of the vector
<code>sdist</code>	The number of standard deviations beyond which values should be down-weighted
<code>taper</code>	A number indicating how strongly values exceeding the standard deviation should taper off
<code>scale</code>	How the weight vector should be scaled: "norm" sets the sum to 1, "max" sets the maximum to 1.
<code>min</code>	A minimum weight.

**Value**

A numeric vector of weights

---

OLcrunch	<i>Crunch Outliers</i>
----------	------------------------

---

**Description**

Crunch Outliers

**Usage**

```
OLcrunch(x, DS = 3, hardlimit = NULL)
```

**Arguments**

<code>x</code>	Numeric vector to remove outliers from
<code>DS</code>	A positive numeric value. If value exceeds this many standard deviations, it is counted as an outlier
<code>hardlimit</code>	A numeric vector with two values. If set, values below the first value and above the second will be counted as outliers, and means/standard deviations will be computed from values within these bounds only.

**Value**

Vector with outlying values set to NA

---

pair	<i>Create unique pairs</i>
------	----------------------------

---

**Description**

Combines vectors such that unique unordered sets are derived from the vectors' cross sections.

**Usage**

```
pair(...)
```

**Arguments**

... two or more vectors of equal length

**Value**

a character vector consisting of all input vectors concatenated term-by-term and in alphabetic order.

**Examples**

```
pair(1:4,4:1)
#[1] "1-4" "2-3" "2-3" "1-4"
```

---

read.csv.folder	<i>Read and merge all .csv files in a folder</i>
-----------------	--

---

**Description**

Read and merge all .csv files in a folder

**Usage**

```
read.csv.folder(
  folder = "./",
  readfunc = list(read.csv, read.csv2, read.table)
)
```

**Arguments**

folder	path to a folder
readfunc	list of functions that will be used to read the files; if the first function fails, the second function will be used, etc.

**Value**

A data.frame containing all merged .csv files

---

RemoveTopTerms	<i>Remove all possible models with one unmoderated term removed</i>
----------------	---

---

### Description

Remove all possible models with one unmoderated term removed

### Usage

```
RemoveTopTerms(form, randeff = "")
```

### Arguments

form	A formula
randeff	The name of the group from which unmoderated terms should be removed. To remove from fixed effects, use "" (the default).

### Value

A list of formulas which have one unmoderated term removed each. The name of each list item is the term which was removed.

### Examples

```
RemoveTopTerms(a ~ b * c + d + (1|e))
#$d
#a ~ b + c + b:c + (1 | e)
#b`b:c`
#a ~ b + c + d + (1 | e)
```

---

retype	<i>Change classes of columns in a data.frame</i>
--------	--

---

### Description

retype() changes the class of specific columns; retype\_all() changes the class of all columns of a given class.

### Usage

```
retype(df, ...)
```

```
retype_all(df, from, to)
```

**Arguments**

<code>df</code>	A <code>data.frame</code>
<code>...</code>	Unquoted column names, paired with the desired class, e.g. <code>age = numeric(), language = character()</code>
<code>from</code>	An empty vector of the class to convert from, or a string. Columns sharing the class of argument <code>from</code> will be converted to the class of argument <code>to</code> .
<code>to</code>	An empty vector of the class to convert to, or a string. Columns sharing the class of argument <code>from</code> will be converted to the class of argument <code>to</code> .

**Examples**

```
sapply(ToothGrowth, class)
#   len      supp    dose
# "numeric" "factor" "numeric"
NewToothGrowth <- retype(ToothGrowth, supp = character(), dose = factor())
sapply(NewToothGrowth, class)
#   len      supp    dose
# "numeric" "character" "factor"

sapply(mtcars, class)
#   mpg      cyl    disp      hp      drat      wt
# "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
#   qsec      vs      am      gear      carb
# "numeric" "numeric" "numeric" "numeric" "numeric"

newmtcars <- retype_all(mtcars, "numeric", "character")
sapply(newmtcars, class)
#   mpg      cyl    disp      hp      drat
# "character" "character" "character" "character" "character"
#   wt      qsec      vs      am      gear      carb
# "character" "character" "character" "character" "character" "character"
```

smoothvect

*Smooth a numeric vector using a moving window algorithm***Description**

Smooth a numeric vector using a moving window algorithm

**Usage**

```
smoothvect(vect, width = 2, both.sides = T, alg = c("mean", "gauss"))
```

**Arguments**

<code>vect</code>	
<code>width</code>	Over how many values should the vector be averaged?
<code>both.sides</code>	If TRUE (default), takes the mean of <code>width</code> values before and after the current index. If FALSE, only takes values ahead of the current index.

**Value**

Smoothed numeric vector

**Examples**

```
temp<- smoothvect(beaver1$temp)
plot(temp,type="l")
```

---

**splitColumn**

*Split a character column into multiple values*

---

**Description**

Split a character column into multiple values

**Usage**

```
splitColumn(x, sep = ";")
```

**Arguments**

x                    a character vector to split into columns  
 sep                  a character separating the different values

**Value**

a `data.frame` of boolean values, with rows representing the unpacked vector entries and columns indicating whether the specific value

---

**theme.pecher**

*Pecher theme for ggplot Based on the plot design style of prof. Diane Pecher.*

---

**Description**

Pecher theme for ggplot Based on the plot design style of prof. Diane Pecher.

**Usage**

```
theme_pecher()
```

---

tokens\_compound\_stepwise

*Compound tokens without overflowing memory and crashing R*


---

## Description

A wrapper around [tokens\\_compound](#) that processes your tokens in chunks, set by argument `stepsize`. See [tokens\\_compound](#) for more info.

## Usage

```
tokens_compound_stepwise(
  x,
  pattern,
  stepsize = 100,
  concatenator = "_",
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  join = TRUE
)
```

---

TransformPlots

*Title*


---

## Description

Visualize how different transformations of the data will fit to a normal distribution.

## Usage

```
TransformPlots(x)
```

## Arguments

`x`                      A numeric vector.

## Examples

```
TransformPlots(mtcars$displ)
```

---

trypackages	<i>Install packages if neccesary, then load them.</i>
-------------	---

---

**Description**

Install packages if neccesary, then load them.

**Usage**

```
trypackages(...)
```

**Arguments**

...	Unquoted names of packages to try loading, and if unable, install and load.
-----	---

**Examples**

```
trypackages(stats,utils,compiler)
```

---

verify_types	<i>Verify variable types in bulk</i>
--------------	--------------------------------------

---

**Description**

Verify variable types in bulk

**Usage**

```
verify_types(...)
```

**Arguments**

...	Named arguments, where the argument is the object to be checked and the name of the argument is the mode (numeric, list, character, etc)
-----	--

**Value**

Returns true on success, causes error if not.



---

wtd.median	<i>Weighted Median</i>
------------	------------------------

---

**Description**

Weighted Median

**Usage**

```
wtd.median(x, wts, na.rm = T)
```

**Arguments**

x	an input vector
wts	a vector of weights
na.rm	Logical indicating whether NA values in the input and weight vectors should be stripped.

**Value**

A weighted median of the input values and weights.

# Index

AnovaTable, [2](#)

clamp, [3](#)  
coerce, [3](#)  
colVars, [4](#)  
combobulate, [4](#)  
compcorr, [5](#)  
CorrCrunch, [5](#)  
CorTable, [6](#)

df.init, [7](#)

ExpandFormula, [7](#)  
ExtractRandomTerms, [8](#)

FindTopTerms, [8](#)

LevenshteinDistance, [9](#)  
logit.weightfun, [9](#)

OLcrunch, [10](#)

pair, [11](#)  
print.AnovaTable (AnovaTable), [2](#)

read.csv.folder, [11](#)  
RemoveTopTerms, [12](#)  
retype, [12](#)  
retype\_all (retype), [12](#)  
rowVars (colVars), [4](#)

smoothvect, [13](#)  
splitColumn, [14](#)

theme\_pecher, [14](#)  
tokens\_compound, [15](#)  
tokens\_compound\_stepwise, [15](#)  
TransformPlots, [15](#)  
trypackages, [16](#)

verify\_types, [16](#)

wtd.median, [17](#)