

# Package ‘eatTools’

November 10, 2022

**Type** Package

**Title** Miscellaneous Functions for the Analysis of Educational Assessments

**Version** 0.7.4

**Depends** R (>= 4.0.0)

**Imports** stats, data.table

**Description** Miscellaneous functions for data cleaning and data analysis of educational assessments. Includes functions for descriptive analyses, character vector manipulations and weighted statistics. Mainly a lightweight dependency for the packages 'eatRep', 'eatGADS', 'eatPrep' and 'eatModel' (which will be subsequently submitted to 'CRAN'). The function for defining (weighted) contrasts in weighted effect coding refers to te Grotenhuis et al. (2017) <[doi:10.1007/s00038-016-0901-1](https://doi.org/10.1007/s00038-016-0901-1)>. Functions for weighted statistics refer to Wolter (2007) <[doi:10.1007/978-0-387-35099-8](https://doi.org/10.1007/978-0-387-35099-8)>.

**License** GPL (>= 2)

**URL** <https://github.com/weirichs/eatTools>

**Suggests** testthat, covr

**NeedsCompilation** no

**Author** Sebastian Weirich [aut, cre],  
Martin Hecht [aut],  
Karoline Sachse [aut],  
Benjamin Becker [aut],  
Nicole Mahler [aut]

**Maintainer** Sebastian Weirich <[sebastian.weirich@iqb.hu-berlin.de](mailto:sebastian.weirich@iqb.hu-berlin.de)>

**Repository** CRAN

**Date/Publication** 2022-11-10 17:00:23 UTC

## R topics documented:

eatTools-package . . . . . 2

addLeadingZerosToCharInt . . . . .	3
asNumericIfPossible . . . . .	4
catch_asNumericIfPossible . . . . .	5
contr.wec.weighted . . . . .	6
crop . . . . .	7
descr . . . . .	8
do_call_rbind_withName . . . . .	9
existsBackgroundVariables . . . . .	10
facToChar . . . . .	10
gsubAll . . . . .	11
halveString . . . . .	12
insert.col . . . . .	13
makeDataFrame . . . . .	13
mergeAttr . . . . .	14
multiseq . . . . .	16
na_omit_selection . . . . .	16
num.to.cat . . . . .	17
print_and_capture . . . . .	18
pwc . . . . .	18
rbind_common . . . . .	19
readMultisep . . . . .	20
recodeLookup . . . . .	20
removeNonNumeric . . . . .	21
removeNumeric . . . . .	22
removePattern . . . . .	22
roundDF . . . . .	23
set.col.type . . . . .	23
tablePattern . . . . .	25
tableUnlist . . . . .	26
whereAre . . . . .	26
wideToLong . . . . .	27
wtdTable . . . . .	28
wtdVar . . . . .	29
<b>Index</b>	<b>30</b>

---

eatTools-package	<i>eatTools: Miscellaneous Functions for the Analysis of Educational Assessments</i>
------------------	--

---

## Description

The eatTools package provides various groups of functions. The main groups of functions include: transformation of vector types, modification of character variables, descriptive analyses and weighted statistics. The package's purpose is mainly to function as a lightweight dependency for other packages.

### Transformation of vector types

The functions `asNumericIfPossible` and `catch_asNumericIfPossible` transform character and factor variables to numeric. `facToChar` transforms factor variables to character. `set.col.type` allows manually setting the type of multiple variables within a `data.frame`.

### Modification of character variables

Multiple convenience functions exist for modification of character variables: removing certain pattern (`removePattern`), removing numerics (`removeNumeric`) and removing non numerics (`removeNonNumeric`), substituting multiple patterns within a string (`gsubAll`) and splitting strings into multiple or a fixed number of parts but at specific position (`halveString`)

### Descriptive Statistics

The function `descr` provides simple descriptive statistics for a `data.frame`, but in a format especially useful for further automated processing (long format `data.frame`).

### Weighted Statistics

`wtdVar` provides calculation of weighted variances (this can be done also by the package `Hmisc`, which has, however, a very high number of dependencies). `wtdTable` provides a weighted frequency table.

---

```
addLeadingZerosToCharInt
```

*Add leading zeros to all columns that can be identified as integers in a character data.frame*

---

### Description

Adds leading zeros to all columns that can be identified as integers in a `data.frame` that consists of character columns only.

### Usage

```
addLeadingZerosToCharInt(dat)
```

### Arguments

`dat` a `data.frame` consisting of character columns only

### Value

a `data.frame` of only character columns and the same dimensions as the input `data.frame` where all columns with integers are all of the same arity now due to added leading zeros.

**Author(s)**

Karoline Sachse

**Examples**

```
dat <- data.frame(v1 = c("0", "300", "e", NA),
                 v2=c("0", "90", "10000", NA),
                 v3=c("k", "kk", "kkk", NA),
                 v4=NA,
                 v5=c("0", "90", "100", "1"))
dat <- set.col.type(dat)
addLeadingZerosToCharInt(dat)
```

---

asNumericIfPossible     *Convert a Vector, Matrix or Data Frame Into Numeric Values If Possible*

---

**Description**

This function converts vectors and matrices of all kinds to numeric. The function can also be used to convert all columns of a data.frame to class numeric for which this conversion is possible i.e. without creating NA when it fails. Non-convertible columns are maintained.

**Usage**

```
asNumericIfPossible(x, maintain.factor.scores = TRUE, force.string = TRUE,
transform.factors = TRUE, varName = NULL)
```

**Arguments**

x	A vector or data frame which should be converted.
maintain.factor.scores	Logical: If TRUE, conversion of the factor levels is attempted (like in as.numeric(as.character(f))). If FALSE, the internal codes of the factor are returned (like in as.numeric(f)). See 'Details'. This argument is only evaluated if transform.factors = TRUE.
force.string	Logical indicating whether columns should be force to numeric, even if NAs are induced. If FALSE, affected columns are maintained. If TRUE, conversion is forced.
transform.factors	Logical indicating whether columns of class factor should be converted. If FALSE, columns of class factor are maintained. If TRUE, conversion of factors is attempted.
varName	Optional: Name of the corresponding variable. Doesn't have to be changed by user.

**Details**

In R, factors may represent ordered categories or categorical variables. Depending on the meaning of the variable, a conversion of the nominal values (of a factor variable) to numeric values may be desirable or not. The arguments `transform.factors` and `maintain.factor.scores` specify if and how factor variables should be treated. See examples.

**Author(s)**

Sebastian Weirich, Karoline Sachse, Benjamin Becker

**Examples**

```
dat <- data.frame(X1 = c("1",NA,"0"), X2 = c("a",NA,"b"),
                 X3 = c(TRUE,FALSE,FALSE), X4 = as.factor(c("a",NA,"b")),
                 X5 = as.factor(c("5","6","7")), stringsAsFactors = FALSE)

str(dat)
asNumericIfPossible(dat)
asNumericIfPossible(dat, transform.factors=TRUE,
                    maintain.factor.scores=FALSE)
asNumericIfPossible(dat, transform.factors=TRUE,
                    maintain.factor.scores=TRUE)
```

---

catch\_asNumericIfPossible

*Use asNumericIfPossible with modified warning.*

---

**Description**

This function uses `asNumericIfPossible` but lets the user change the warning issued by `asNumericIfPossible`. Suited for use in other R packages.

**Usage**

```
catch_asNumericIfPossible(x, warn, maintain.factor.scores = TRUE,
                          force.string = TRUE, transform.factors = TRUE)
```

**Arguments**

<code>x</code>	A vector or data frame which should be converted.
<code>warn</code>	A character vector of length 1 with the desired warning.
<code>maintain.factor.scores</code>	Logical: If TRUE, conversion of the factor levels is attempted (like in <code>as.numeric(as.character(f))</code> ). If FALSE, the internal codes of the factor are returned (like in <code>as.numeric(f)</code> ). See 'Details'. This argument is only evaluated if <code>transform.factors = TRUE</code> .
<code>force.string</code>	Logical indicating whether columns should be force to numeric, even if NAs are induced. If FALSE, affected columns are maintained. If TRUE, conversion is forced.

transform.factors

Logical indicating whether columns of class factor should be converted. If FALSE, columns of class factor are maintained. If TRUE, conversion of factors is attempted.

## Details

For details see [asNumericIfPossible](#)

## Author(s)

Benjamin Becker

## Examples

```
char <- c("a", "b", 1)
catch_asNumericIfPossible(x = char, warn = "Vector could not be converted")
```

---

contr.wec.weighted	<i>Calculates contrasts for a weighted factor variable based on weighted effect coding</i>
--------------------	--

---

## Description

Function works equivalent to `contr.wec` from the `wec` package, but allows for weighted contrasts.

## Usage

```
contr.wec.weighted(x, omitted, weights)
```

## Arguments

x	grouping variable of class factor
omitted	Label of the factor label that should be taken as the omitted category
weights	Numeric vector of non-negative weights

## Value

Returns a contrast matrix based on weighted effect coding.

## Author(s)

Sebastian Weirich, based upon the `contr.wec` function of the `wec` package

## Examples

```
### exemplary data according to wec paper
dat <- data.frame ( group = as.factor(c(rep(1,3), rep(2,2))), wgt = c(2/3, 4/3, 2, 3/8, 5/8))
### default contrasts
contrasts(dat[, "group"])
### weighted effect coding for weighted data
contr.wec.weighted(x= dat[, "group"], omitted=1, weights=dat[, "wgt"])
### equal to weighted effect coding: wec::contr.wec(x= dat[, "group"], omitted=1)
contr.wec.weighted(x= dat[, "group"], omitted=1, weights=rep(1, nrow(dat)))
```

---

crop

*Remove Trailing and Leading Characters From Character Strings*

---

## Description

Similarly to the function `trim` from the `gdata` package, this function can be used to remove trailing and leading spaces from character strings. However, in contrast to `trim`, any character can be removed by `crop`.

## Usage

```
crop(x, char = " ")
```

## Arguments

x	character string
char	character to be removed from beginning and end of x

## Author(s)

Martin Hecht, Sebastian Weirich

## Examples

```
str <- c(" 12 kk ", "op j   q ", "110")
crop(str)
crop(str, "op")
```

---

descr *Descriptive statistics for one or several variables*

---

### Description

Function computes descriptive statistics for one variable or several variables within a data frame.

### Usage

```
descr (variable, na = NA, p.weights = NULL, na.rm = FALSE, verbose=TRUE)
```

### Arguments

variable	one variable or a data.frame with several variables
na	optional values with should be considered a missing values
p.weights	optional: vector with individual weights if weighted statistics should be computed
na.rm	logical: should missings be removed prior to estimation?
verbose	logical: Print messages to console?

### Value

a data frame with the following columns

N	number of observations
N.valid	number of non-missing observations
Missing	number of missings
Minimum	minimum of numeric variables
Maximum	maximum of numeric variables
Sum	sum of numeric variables
Mean	arithmetic mean of numeric variables
std.err	standard error of the arithmetic mean. Note: for weighted means, standard error is estimated according to Cochran (1977): $\sigma_x^2 = n / ((n - 1) * w_s^2) * \text{Sigma}(w_i^2 * (x_i - x))$ .
sig	p value
Median	median of numeric variables
SD	standard deviation of numeric variables
Var	variance of numeric variables

### Author(s)

Sebastian Weirich



## References

Cochran W. G. (1977). *Sampling Techniques* (3rd Edn). Wiley, New York

## Examples

```
data(mtcars)
descr(mtcars)
```

---

do\_call\_rbind\_withName

*Row bind a list while assigning names to rows*

---

## Description

Use `do.call(rbind, ...)` on a list of `data.frames` while creating a new variable (`colName`) which contains, for example, the original list naming (`name`).

## Usage

```
do_call_rbind_withName(df_list, name = names(df_list), colName)
```

## Arguments

<code>df_list</code>	A list of <code>data.frames</code> .
<code>name</code>	Vector of names to fill <code>colName</code> . Default uses the names of <code>df_list</code> .
<code>colName</code>	A single character; name for the new column.

## Value

Returns a `data.frame`.

## Author(s)

Benjamin Becker

## Examples

```
### create example list
df_list <- lapply(mtcars, function(x) {
  data.frame(m = mean(x), sd = sd(x))
})

### transform to a single data.frame
do_call_rbind_withName(df_list, colName = "variable")
```

---

existsBackgroundVariables

*Internally needed function for consistency checks and data preparation.*

---

**Description**

Function is necessary for eatRep and eatModel as well and therefore exported to namespace.

**Usage**

```
existsBackgroundVariables (dat, variable, warnIfMissing = FALSE)
```

**Arguments**

dat	A data frame
variable	column number or variable name
warnIfMissing	Logical: gives a warning if the variable contains missing values

**Value**

a structured list of variable names

**Author(s)**

Sebastian Weirich

**Examples**

```
data(mtcars)
existsBackgroundVariables(mtcars, 2:4)
```

---

facToChar

*Transform columns in a data frame*

---

**Description**

Function transforms all data frame columns of a specific class into another class.

**Usage**

```
facToChar ( dataFrame, from = "factor", to = "character")
```

**Arguments**

dataFrame	a data frame
from	which column class should be transformed?
to	target column class

**Value**

a data frame

**Author(s)**

Sebastian Weirich

**Examples**

```
data(mtcars)
### original classes
sapply(mtcars, class)
mtcars1 <- facToChar(mtcars, from = "numeric", to = "character")
sapply(mtcars1, class)
```

---

gsubAll

*Pattern matching and replacement*

---

**Description**

Function is a wrapper for gsub() which allows to replace more than one pattern.

**Usage**

```
gsubAll ( string, old, new)
```

**Arguments**

string	a character vector where matches are sought
old	character vector containing strings to be matched in the given character vector named string.
new	a replacement for matched pattern

**Value**

character vector with replaced patterns

**Author(s)**

Benjamin Becker

**Examples**

```
### replace all numbers by words
txt <- "1 example for 2 reasons in 4 seasons"
gsubAll ( txt, old = as.character(1:4), new = c("one", "two", "three", "four"))
```

---

halveString

*Split string exactly in two parts*


---

**Description**

strsplit splits a string according to a specific sign. The number of occurrences of the splitting sign defines the number of splits. halveString allows to split the string in only two parts, no matter how often the splitting sign occurs.

**Usage**

```
halveString (string, pattern, first = TRUE , colnames=c("X1", "X2"))
```

**Arguments**

string	A character vector.
pattern	character vector (or object which can be coerced to such) to use for splitting.
first	Logical: Relevant if the pattern occurs more than one time in the string. Defines whether the first (default) or last occurrence is used for splitting.
colnames	Optional: character vector of length 2 to specify the colnames of the resulting data.frame.

**Value**

A data.frame with two columns

**Examples**

```
str <- c("John_Bolton", "Richard_Milhouse_Nixon", "Madonna")
strsplit(str, split = "_")
halveString(str, pattern = "_")
halveString(str, pattern = "_", first=FALSE)
```

---

insert.col	<i>Insert Columns into a Data Frame in a Specific Position</i>
------------	--

---

**Description**

Insert columns into a data frame in specific position

**Usage**

```
insert.col(dat, toinsert, after)
```

**Arguments**

dat	A data frame
toinsert	Column name(s) or column number(s) of the columns to be reinserted
after	Column name or column number after which the columns specified in insert should be reinserted.

**Value**

A data frame with columns in specified positions.

---

makeDataFrame	<i>Converts tbl or data.table objects to plain data.frames for internal processing</i>
---------------	--

---

**Description**

Function is mainly used for internal checks in the eatRep and eatModel package: objects which expected to be data.frames for further processing are converted to data.frame when their class is tbl, for example.

**Usage**

```
makeDataFrame (dat, name = "dat")
```

**Arguments**

dat	An object which is intended to be a data.frame.
name	Optional: name of data.frame for use in messages

**Value**

data frame.

## Examples

```
dat <- data.table::data.table(x1 = 1:5, y1 = letters[1:5])
# unexpected in 'classical' data frames
class(dat[, "x1"])
dat <- makeDataFrame(dat)
```

---

mergeAttr	<i>Merge Two Data Frames with additional messages and maintain variable attributes</i>
-----------	--

---

## Description

This is a wrapper for the [merge](#) function. `merge` does not maintain variable attributes. `mergeAttr` might be useful if variable attributes should be maintained. For example, if SPSS data are imported via [read.spss](#), variable and value labels are stored as attributes which get lost if data are merged subsequently. Moreover, function gives additional messages if (combination of) by-variables are not unique in at least one data.frame, or if by-variables have different classes, or if some units of the by-variables are missing in one of the data sets. Users are free to specify which kind of messages are desirable.

## Usage

```
mergeAttr(x, y, by = intersect(names(x), names(y)),
  by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
  sort = TRUE, suffixes = c(".x", ".y"), setAttr = TRUE, onlyVarVallabs = TRUE,
  homoClass = TRUE, unitName = "unit", xName = "x", yName = "y",
  verbose = c("match", "unique", "class", "dataframe"))
```

## Arguments

<code>x</code>	first data frame to be merged.
<code>y</code>	second data frame to be merged.
<code>by</code>	specifications of the columns used for merging
<code>by.x</code>	specifications of the columns used for merging
<code>by.y</code>	specifications of the columns used for merging
<code>all</code>	logical; <code>all = L</code> is shorthand for <code>all.x = L</code> and <code>all.y = L</code> , where <code>L</code> is either <code>TRUE</code> or <code>FALSE</code> .
<code>all.x</code>	logical; if <code>TRUE</code> , then extra rows will be added to the output, one for each row in <code>x</code> that has no matching row in <code>y</code> . These rows will have NAs in those columns that are usually filled with values from <code>y</code> . The default is <code>FALSE</code> , so that only rows with data from both <code>x</code> and <code>y</code> are included in the output.
<code>all.y</code>	logical; analogous to <code>all.x</code> .
<code>sort</code>	logical. Should the result be sorted on the by columns?

suffixes	a character vector of length 2 specifying the suffixes to be used for making unique the names of columns in the result which not used for merging (appearing in by etc).
setAttr	Logical: restore the variable attributes? If FALSE, the behavior of mergeAttr equals the behavior of merge.
onlyVarValLabs	Logical: If TRUE, only the variable and value labels as captured by read.spss and stored by convertLabel from the eatAnalysis package will be restored. If FALSE, all variable attributes will be restored.
homoClass	Logical: Beginning with R version 3.5, merge may give an error if the class of the by-variables differs in both data.frames. If TRUE, class of by-variable(s) will be homogenized before merging.
unitName	Optional: Set the name for the unit variable to get more informative messages. This is mainly relevant if mergeAttr is called from other functions.
xName	Optional: Set the name for the x data.frame to get more informative messages. This is mainly relevant if mergeAttr is called from other functions.
yName	Optional: Set the name for the y data.frame to get more informative messages. This is mainly relevant if mergeAttr is called from other functions.
verbose	Optional: Choose whether messages concerning missing levels in by-variables should be printed on console ("match"), or messages concerning uniqueness of by-variables ("unique"), or messages concerning different classes of by-variables ("class"), or messages concerning appropriate class (data.frame) of x and y ("dataframe"). Multiple choices are possible, e.g. verbose = c("match", "class"). If verbose = TRUE, all messages are printed, if verbose = FALSE, no messages are printed at all. The default is equivalent to verbose = TRUE.

## Value

data frame. See the help page of [merge](#) for further details.

## Examples

```
### data frame 1, variable 'y' with variable.label 'test participation'
df1 <- data.frame ( id = 1:3, sex = factor ( c("male", "male", "female")),
  happy = c("low", "low", "medium"))
attr(df1[,"happy"], "variable.label") <- "happieness in the workplace"

### data frame 2 without labels
df2 <- data.frame ( id = as.factor(c(2,2,4)), status = factor ( c("married", "married", "single")),
  convicted = c(FALSE, FALSE, TRUE))

### lost label after merging
df3 <- merge(df1, df2, all = TRUE)
attr(df3[,"happy"], "variable.label")

### maintain label
df4 <- mergeAttr(df1, df2, all = TRUE, onlyVarValLabs = FALSE)
attr(df4[,"happy"], "variable.label")
```

```
### adapt messages
df5 <- mergeAttr(df1, df2, all = TRUE, onlyVarValLabs = FALSE, unitName = "student",
  xName = "student questionnaire", yName = "school questionnaire",
  verbose = c("match", "unique"))
```

---

multiseq                      *multiple sequences*

---

### Description

creates a sequence for every unique value in a vector

### Usage

```
multiseq(v)
```

### Arguments

v                      a vector

### Value

a vector with multiple sequences

### Author(s)

Martin Hecht

### Examples

```
v <- c("a", "a", "a", "c", "b", "b", "a")
multiseq(v)
```

---

na\_omit\_selection            *Drop rows containing missing values*

---

### Description

Drop rows containing missing values in selected columns.

### Usage

```
na_omit_selection (dat, varsToOmitIfNA)
```



**Arguments**

`dat` a data.frame  
`varsToOmitIfNA` Name or column number of the variables which should be considered for row deletion due to NAs

**Value**

A data.frame with deleted rows

**Examples**

```
dat1 <- data.frame ( v1 = c(1,NA,3), v2 = c(letters[1:2],NA),
                    v3 = c(NA, NA, TRUE), stringsAsFactors = FALSE)
na.omit(dat1)
na_omit_selection(dat1, "v2")
```

---

num. to. cat	<i>Transform continuous variables into ordered factors</i>
--------------	--

---

**Description**

Function is useful if parameters on the ‘PISA’ metric should be transformed into competence levels.

**Usage**

```
num.to.cat(x, cut.points, cat.values = NULL)
```

**Arguments**

`x` Numeric vector.  
`cut.points` Numeric vector with cut scores.  
`cat.values` Optional: vector with labels for the cut scores. Note: if specified, length of `cat.values` should be `length(cut.points)+1`.

**Value**

Vector with factor values.

**Author(s)**

Sebastian Weirich

**Examples**

```
values <- rnorm(10,0,1.5) * 100 + 500
num.to.cat(x = values, cut.points = 390+0:3*75)
num.to.cat(x = values, cut.points = 390+0:3*75, cat.values = c("1a", "1b", 2:4))
```

---

print\_and\_capture      *Easy integration of (small) tables into (error) messages*

---

### Description

Some (error) messages are more understandable if small (frequency) tables are used for clearness. The function simplifies integration of these tables. The function is intended to be used in combination with `message`, `stop`, or `cat`, for example.

### Usage

```
print_and_capture (x, spaces = 0)
```

### Arguments

`x`                      The object which should be integrated. Normally, a (small) table or data frame.  
`spaces`                 Number of spaces between left border and the table

### Value

a string which may be combined with messages

### Examples

```
frequency.table <- as.table(matrix(c(12,0,5,7),2,2))
attr(frequency.table, "dimnames") <- list("sex" = c("male", "female"),
      "migration" = c(TRUE, FALSE))
message("Some combinations of variables with zero observations: \n",
      print_and_capture(frequency.table, spaces = 5))
```

---

pwc                      *Part-whole correlation for numeric data frames*

---

### Description

Computes the part-whole correlation (correlation of an item with the whole scale except for this item)

### Usage

```
pwc(dat)
```

### Arguments

`dat`                     a data.frame with numeric columns (items)

**Value**

A data.frame with three columns: First column item identifier, second column with conventional item-scale correlation, third column with part-whole correlation

**Examples**

```
dat <- data.frame ( item1 = c(0,1,1,3), item2 = c(2,3,1,3), item3 = c(1, NA, 3,3))
pwc(dat)
```

---

rbind\_common

*Combine data.frames by row, using only common columns.*

---

**Description**

rbinds a list of data.frames, using only these columns which occur in each of the single data.frames.

**Usage**

```
rbind_common(...)
```

**Arguments**

... input data frames to row bind together. The first argument can be a list of data frames, in which case all other arguments are ignored. Any NULL inputs are silently dropped. If all inputs are NULL, the output is NULL. If the data.frames have no common columns, the output is NULL and a warning is given.

**Value**

a single data frame

**Examples**

```
### data frame 1
df1 <- data.frame ( a = 1:3, b = TRUE)

### data frame 2
df2 <- data.frame ( d = 100, a = 11:13)

### data frame 3
df3 <- data.frame ( d = 1000, x = 101:103)

### one common col
rbind_common(df1, df2)

### no common cols
rbind_common(df1, df2, df3)
```

---

readMultisep	<i>Read in data.frames with separator characters &gt;=1Byte</i>
--------------	---

---

**Description**

Read in character separated data.frames with separator characters >=1Byte.

**Usage**

```
readMultisep(file, sep)
```

**Arguments**

file	the name of the file which the data are to be read from.
sep	the field separator character(s).

**Value**

A data frame containing a representation of the data in the file.

**Examples**

```
filePath <- tempfile(fileext = ".txt")
dat <- data.frame(v1 = c("0", "300", "e", NA),
                 v2 = c("0", "90", "10000", NA),
                 v3 = c("k", "kk", "kkk", NA),
                 v4 = NA,
                 v5 = c("0", "90", "100", "1"))
write.table(dat, file = filePath, row.names = FALSE, col.names = FALSE, sep = "]&")
readMultisep(filePath, sep = "]&")
```

---

recodeLookup	<i>Recode a variable according to a lookup table</i>
--------------	--

---

**Description**

Recodes the values of a variable. Function resembles the recode function from the car package, but uses a lookup table to specify old and new values.

**Usage**

```
recodeLookup(var, lookup)
```

**Arguments**

`var` a vector (e.g. numeric, character, or factor)  
`lookup` a data.frame with exact two columns. First column contains old values, second column new values. Values which do not occur in the old column remain unchanged.

**Value**

a vector of the same length as `var` with recoded values

**Examples**

```
num_var <- sample(1:10, size = 10, replace = TRUE)
lookup <- data.frame(old = c(2, 4, 6), new = c(200,400,600))
num_var2<- recodeLookup(num_var, lookup)
```

---

`removeNonNumeric` *Removes all non-numeric characters from a string.*

---

**Description**

Function removes all non-numeric characters from a string.

**Usage**

```
removeNonNumeric ( string)
```

**Arguments**

`string` a character vector

**Value**

a character string

**Author(s)**

Sebastian Weirich

**Examples**

```
str <- c(".d1.nh.120", "empty", "110", ".nh.dgd", "only.nh")
removeNonNumeric(str)
```

---

removeNumeric	<i>Removes alphanumeric characters from a string.</i>
---------------	---

---

**Description**

Function removes alphanumeric characters from a string.

**Usage**

```
removeNumeric ( string)
```

**Arguments**

string	a character vector
--------	--------------------

**Value**

a character string

**Author(s)**

Sebastian Weirich

**Examples**

```
str <- c(".d1.nh.120", "empty", "110", ".nh.dgd", "only.nh")
removeNumeric(str)
```

---

removePattern	<i>Removes a specified pattern from a string.</i>
---------------	---

---

**Description**

Function remove a specified string from a character vector.

**Usage**

```
removePattern ( string, pattern)
```

**Arguments**

string	a character vector
pattern	a character pattern

**Value**

a character string

**Author(s)**

Sebastian Weirich

**Examples**

```
str <- c(".d1.nh.120", "empty", "110", ".nh.dgd", "only.nh")
removePattern(str, ".nh.")
```

---

roundDF	<i>Round a data.frame.</i>
---------	----------------------------

---

**Description**

Round all numeric variables in a `data.frame`, leave the other variables untouched. Column and row names are preserved.

**Usage**

```
roundDF(dat, digits = 3)
```

**Arguments**

<code>dat</code>	A <code>data.frame</code> .
<code>digits</code>	Integer indicating the number of decimal places.

**Value**

Returns the rounded `data.frame`.

**Examples**

```
roundDF(mtcars, digits = 0)
```

---

set.col.type	<i>Set the Class of Columns in a Data Frame</i>
--------------	---

---

**Description**

This function converts the classes of columns to character, numeric, logical, integer or factor.

**Usage**

```
set.col.type(dat, col.type = list("character" = NULL), verbose = FALSE, ...)
```

**Arguments**

<code>dat</code>	A data frame
<code>col.type</code>	A named list of column names that are to be converted. The names of the list indicate the class to which the respective column should be converted (character, numeric, numeric.if.possible, logical, integer or factor)
<code>verbose</code>	if TRUE details about converted columns are printed on the console
<code>...</code>	Additional arguments to be passed to <code>asNumericIfPossible</code>

**Details**

Use `col.type="numeric.if.possible"` if conversion to numeric should be tested upfront, see `asNumericIfPossible` for details.

**Value**

A data frame with column classes changed according to the specifications in `col.type`

**Author(s)**

Martin Hecht, Karoline Sachse

**See Also**

`asNumericIfPossible`

**Examples**

```
str(d <- data.frame("var1" = 1, "var2" = TRUE, "var3" = FALSE,
  "var4" = as.factor(1), "var5" = as.factor("a"), "var6" = "b",
  stringsAsFactors = FALSE))

str(set.col.type(d))
str(set.col.type(d, list("numeric" = NULL)))
str(set.col.type(d, list("character" = c("var1", "var2"),
  "numeric" = "var3", "logical" = "var4")))
str(set.col.type(d, list("numeric.if.possible" = NULL)))
str(set.col.type(d, list("numeric.if.possible" = NULL),
  transform.factors = TRUE))
str(set.col.type(d, list("numeric.if.possible" = NULL), transform.factors = TRUE,
  maintain.factor.scores = FALSE))
```



---

tablePattern	<i>Creates skeleton for frequency tables with desired values</i>
--------------	--

---

### Description

Function takes values and creates a frequency table including these values. Models behavior of factor variables.

### Usage

```
tablePattern(x, pattern = NULL, weights, na.rm = TRUE,  
useNA = c("no", "ifany", "always"))
```

### Arguments

x	a vector
pattern	desired values for table output
weights	optional: weights
na.rm	should missing values be removed
useNA	whether to include [NA] values in the table

### Value

a frequency table

### Author(s)

Sebastian Weirich

### Examples

```
grades <- c(1,1,3,4,2,3,4,5,5,3,2,1)  
table(grades)  
tablePattern(grades, pattern = 1:6)
```

---

tableUnlist	<i>Frequency table for data frames, e.g. across multiple columns</i>
-------------	--

---

**Description**

Replaces the somehow buggy function combination `table(unlist(data))`.

**Usage**

```
tableUnlist(dataFrame, useNA = c("no", "ifany",  
  "always"))
```

**Arguments**

dataFrame	Data frame with more than one column.
useNA	whether to include NA values in the table. See help file of <code>table</code> for more details.

**Value**

A frequency table

**Author(s)**

Sebastian Weirich

**Examples**

```
dat <- data.frame ( matrix ( data = sample(0:1,200,replace=TRUE), nrow=20, ncol=10))  
tableUnlist(dat)
```

---

whereAre	<i>Matches a scalar with elements of a vector.</i>
----------	--

---

**Description**

The function closely resembles the `match` function, but allows for multiple matches.

**Usage**

```
whereAre(a,b,verbose=TRUE)
```

**Arguments**

a	a scalar
b	a numeric or character vector
verbose	logical: print messages on console?

**Value**

A numeric vector

**Author(s)**

Sebastian Weirich

**Examples**

```
a <- 12
b <- c(10, 11, 12, 10, 11, 12)
match(a, b)
whereAre(a=a, b=b)
```

---

wideToLong	<i>Transform wide format data sets into the long format necessary for eatRep analyses</i>
------------	---

---

**Description**

Data from large-scale assessments often are provided in the wide format. This function easily transform data into the long format required by eatRep.

**Usage**

```
wideToLong (datWide, noImp, imp, multipleColumns = TRUE, variable.name = "variable",
            value.name = "value")
```

**Arguments**

datWide	Data set in the wide format, i.e. one row per person
noImp	character vector of non-imputed variables which are desired for following analyses
imp	Named list of character vectors which include the imputed variables which are desired for following analyses
multipleColumns	Logical: use one column for each imputed variable (if more than one imputed variable is used)? Alternatively, only one column for all imputed variables is used (this is the default behavior of the melt function from the reshape2 package).
variable.name	Applies only if multipleColumns = "FALSE": name of variable used to store measured variable names
value.name	Applies only if multipleColumns = "FALSE": name of variable used to store values

**Value**

A data.frame in the long format.

**Author(s)**

Sebastian Weirich

**Examples**

```
### create arbitrary wide format large-scale assessment data for two
### subjects, each with three imputations
datWide <- data.frame ( id = paste0("P",1:5), weight = abs(rnorm(5,10,1)),
  country = c("USA", "BRA", "TUR", "GER", "AUS"),
  sex = factor(c("female", "male", "female", "female", "male")),
  matrix(data = rnorm(n=15, mean = 500, sd = 75),
    nrow=5, dimnames = list(NULL, paste0("mat.pv", 1:3))),
  matrix(data = rnorm(n=15, mean = 480, sd = 80),
    nrow=5, dimnames = list(NULL, paste0("sci.pv", 1:3))),
  stringsAsFactors=FALSE)
datLong <- wideToLong(datWide = datWide, noImp = c("id", "weight", "country", "sex"),
  imp = list ( math = paste0("mat.pv", 1:3),
  science = paste0("sci.pv", 1:3)))
datLong2<- wideToLong(datWide = datWide, noImp = c("id", "weight", "country", "sex"),
  imp = list ( math = paste0("mat.pv", 1:3),
  science = paste0("sci.pv", 1:3)),
  multipleColumns = FALSE, variable.name = "varName",
  value.name = "val")
```

---

wtdTable

*Computed weighted frequency tables*

---

**Description**

This functions works quite equally as the wtd.table function from the Hmisc package.

**Usage**

```
wtdTable(x , weights , na.rm = FALSE)
```

**Arguments**

x	a character or category or factor vector
weights	a numeric vector of non-negative weights
na.rm	set to FALSE to suppress checking for NAs. If TRUE, NAs are removed from x as well as from weights prior to variance estimation.

**Value**

a frequency table

**Examples**

```
x <- c(50, 1, 50)
w <- c(1, 4, 1)
wtdTable(x, w)
```

---

wtdVar

*Computed weighted variance*

---

**Description**

This functions works quite equally as the `wtd.var` function from the `Hmisc` package.

**Usage**

```
wtdVar(x , weights , na.rm = FALSE)
```

**Arguments**

<code>x</code>	numeric vector
<code>weights</code>	a numeric vector of non-negative weights
<code>na.rm</code>	set to <code>FALSE</code> to suppress checking for NAs. If <code>TRUE</code> , NAs are removed from <code>x</code> as well as from <code>weights</code> prior to variance estimation.

**Value**

a scalar

**Author(s)**

Benjamin Becker

**Examples**

```
x <- c(50, 1, 25)
w <- c(1, 4, 1)
wtdVar(x, w)
```

# Index

`addLeadingZerosToCharInt`, 3  
`asNumericIfPossible`, 3, 4, 6

`cat`, 18  
`catch_asNumericIfPossible`, 3, 5  
`contr.wec.weighted`, 6  
`crop`, 7

`descr`, 3, 8  
`do_call_rbind_withName`, 9

`eatTools-package`, 2  
`existsBackgroundVariables`, 10

`facToChar`, 3, 10

`gsubAll`, 3, 11

`halveString`, 3, 12

`insert.col`, 13

`makeDataFrame`, 13  
`merge`, 14, 15  
`mergeAttr`, 14  
`message`, 18  
`multiseq`, 16

`na_omit_selection`, 16  
`num.to.cat`, 17

`print_and_capture`, 18  
`pwc`, 18

`rbind_common`, 19  
`read.spss`, 14  
`readMultisep`, 20  
`recodeLookup`, 20  
`removeNonNumeric`, 3, 21  
`removeNumeric`, 3, 22  
`removePattern`, 3, 22  
`roundDF`, 23

`set.col.type`, 3, 23  
`stop`, 18

`tablePattern`, 25  
`tableUnlist`, 26

`whereAre`, 26  
`wideToLong`, 27  
`wtdTable`, 3, 28  
`wtdVar`, 3, 29