

Package ‘eatTools’

September 25, 2026

Type Package

Title Miscellaneous Functions for the Analysis of Educational Assessments

Version 0.7.10

Depends R (>= 4.1)

Imports stats, data.table, stringi, checkmate

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>,
<https://weirichs.github.io/eatTools/>

Suggests testthat, covr

NeedsCompilation no

Config/roxygen2/version 8.1.0

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

Maintainer Sebastian Weirich <sebastian.weirich@iqb.hu-berlin.de>

Repository CRAN

Date/Publication 2026-09-25 14:00:08 UTC

Contents

eatTools-package	3
addLeadingZerosToCharInt	3
asNumericIfPossible	4
catch_asNumericIfPossible	5
checkBackgroundVariables	6
cleanifyString	8
contr.wec.weighted	9
crop	10
data.timss1	10
descr	11
do_call_rbind_withName	12
existsBackgroundVariables	13
facToChar	14
gsubAll	15
halveString	15
insert.col	16
makeDataFrame	17
makeTria	18
mergeAttr	19
multiseq	21
na_omit_selection	22
num.to.cat	23
print_and_capture	23
pwc	24
rbind_common	25
rbind_fill_vector	25
readMultisep	26
recodeLookup	27
removeNonNumeric	27
removeNumeric	28
removePattern	29
roundDF	29
seq2	30
set.col.type	30
tablePattern	31
tableUnlist	32
whereAre	33
wideToLong	33
wtdTable	35
wtdVar	35
%%\$%	36

Index

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. In any column containing strings that can be converted to integers, these strings will be padded with leading zeros so that all values in the column have the same number of digits.

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	<i>Convert a Vector, Matrix or Data Frame Into Numeric Values If Possible</i>
---------------------	---

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

<code>x</code>	A vector or data frame which should be converted.
<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.
<code>transform.factors</code>	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.
<code>varName</code>	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.
<code>transform.factors</code>	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")
```

checkBackgroundVariables

Internally needed function for consistency checks and data preparation.

Description

Function is necessary for `eatRep` and `eatModel` to verify input consistency and therefore exported to namespace.

Usage

```
checkBackgroundVariables (varList, len1 = NULL, overlap = NULL)
```

Arguments

varList	A non-nested list of objects, where each individual object corresponds to the returned object of the existsBackgroundVariables function.
len1	The names of varList which elements must have length 1.
overlap	A list of variable groups which elements must overlap with a given frequency.

Details

The function checks whether certain variables or variable groups that are used for analyses within `eatRep` or `eatModel` fulfill certain properties. For example, there may only be one ID variable, but there must be several item variables (items). Item variables and other covariates must be disjoint, while variables for group differences must be a subset of the group variables.

Value

There are no returns. If the input is inconsistent, the functions aborts with an error message.

Examples

```
## Not run:
# borrow some data from the BIFIESurvey package
data(data.timss1)
varGroups <- list(ID = "IDSTUD", wgt = c("TOTWGT", "JKZONE"),
  covars = c("female", "books", "lang", "migrant"), depVar = "ASMMAT")

# error if one variable does not occur in data due to typo
varList <- lapply(varGroups, FUN = function (v) {
  existsBackgroundVariables (dat=data.timss1[[1]], variable=v)
})

# correction
varGroups <- list(ID = "IDSTUD", wgt = c("TOTWGT", "JKZONE"),
  covars = c("female", "books", "lang", "migrant"), depVar = "ASMMAT")
varList <- lapply(varGroups, FUN = function (v) {
  existsBackgroundVariables (dat=data.timss1[[1]], variable=v)
})

# however, only one weighting variable is allowed
check <- checkBackgroundVariables (varList, len1 = c("ID", "wgt", "depVar"),
  overlap = NULL)

# only one of the covariates should be used for group differences
# difference variable must be included in covars
varGroups <- list(ID = "IDSTUD", wgt = "TOTWGT", covars = c("female", "books", "lang"),
  differencesFor = "migrant", depVar = "ASMMAT")
varList <- lapply(varGroups, FUN = function (v) {
  existsBackgroundVariables (dat=data.timss1[[1]], variable=v)
})
check <- checkBackgroundVariables (varList, len1 = c("ID", "wgt", "depVar"),
  overlap = list(v1 = list(vars = c("covars", "differencesFor"), len = 1)))
```

```
## End(Not run)
```

cleanifyString	<i>Removes special characters</i>
----------------	-----------------------------------

Description

Removes special characters from a character string. Also applicable to factor variables and `data.frame`s.

Usage

```
cleanifyString(x, removeNonAlphaNum = TRUE, replaceSpecialChars = TRUE,
               oldEncoding = NULL, ...)
```

Arguments

<code>x</code>	a character variable, factor variable or <code>data.frame</code>
<code>removeNonAlphaNum</code>	logical. If TRUE, all non-alphanumeric characters are removed. Default is TRUE.
<code>replaceSpecialChars</code>	logical. If TRUE, special characters are replaced. Default is TRUE.
<code>oldEncoding</code>	character. The encoding of the input data if it should be transformed to "UTF-8".
<code>...</code>	further arguments passed to other methods.

Details

If unwanted characters are removed from a character string in a factor variable, this can lead to a change in the factor structure (according to the reference category, for example). `cleanifyString` restores the factor structure after removing special characters. Function is mainly used internally in the `eatRep`, `eatGADS`, and `eatModel` packages.

Value

a character variable, factor variable or `data.frame` with removed special characters

Examples

```
fac1 <- factor(c("Tablet-Paper", "Computer.(Laptop)", "Computer.(Laptop)"),
               levels = c("Tablet-Paper", "Computer.(Laptop)"))
table(fac1)

# Remove special characters
fac2 <- cleanifyString(fac1)
fac2
```

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

<code>x</code>	grouping variable of class factor
<code>omitted</code>	Label of the factor label that should be taken as the omitted category
<code>weights</code>	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

<code>x</code>	character string
<code>char</code>	character to be removed from beginning and end of <code>x</code>

Author(s)

Martin Hecht, Sebastian Weirich

Examples

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

data.timss1

Example dataset TIMSS 2011

Description

The dataset `data.timss1` is borrowed from the `BIFIEsurvey` package. `data.timss1` is a list containing 5 imputed datasets.

Usage

```
data(data.timss1)
```

Format

'list': 5 imputed data sets, each which 4668 observations and 12 variables

- IDSTUD** individual student identifier
- TOTWGT** Total case weight
- JKZONE** jackknifing zone
- JKREP** jackknife replicate
- female** indicator whether student's sex is female
- books** Number of books at home
- lang** language spoken at home
- migrant** student's migration background
- scsci** self concept science
- likesc** pleasure in science
- ASMMAT** student's score in math
- ASSSCI** student's score in science

References

Robitzsch, A., & Oberwimmer, K. (2024). BIFIEsurvey: Tools for survey statistics in educational assessment. R package version 3.6-6, <https://CRAN.R-project.org/package=BIFIEsurvey>.

descr	<i>Descriptive statistics for one or several variables</i>
-------	--

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

df_list	A list of data.frames.
name	Vector of names to fill colName. Default uses the names of df_list.
colName	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,
  stopIfMissingOnVars = NULL)
```

Arguments

dat	A data frame
variable	vector of column number(s) or variable name(s)
warnIfMissing	Logical: gives a warning if the variable contains missing values
stopIfMissingOnVars	Character vector of variable names. Only for these variables, warnings as raised through warnIfMissing = TRUE are turned into errors.

Value

a structured list of variable names

Examples

```
# for more detailed examples, see help page of checkBackgroundVariables
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. Does not allow using regular expressions (internally, `gsub(..., fixed = TRUE)` is used).

Usage

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

Arguments

<code>string</code>	a character vector where matches are sought
<code>old</code>	character vector containing strings to be matched in the given character vector named <code>string</code> . Can only contain unique entries.
<code>new</code>	a replacement for matched pattern

Details

Internally, the function calls `gsub()` repeatedly, beginning with the longest string in `old`. String length is evaluated using `nchar()`. This is done to avoid repeated modifications if strings in `old` match each other.

Value

character vector with replaced patterns

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 regular expression. The number of occurrences of the splitting regular expression defines the number of splits. `halveString` allows to split the string in only two parts, no matter how often the splitting regular expression 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 matrix with two columns

Examples

```
str1 <- c("John_Bolton", "Richard_Milhouse_Nixon", "Madonna")
strsplit(str1, split = "_")
halveString(str1, pattern = "_")
halveString(str1, pattern = "_", first=FALSE)

# split patterns with more than one character and regular expression
str2 <- c("John._.Bolton", "Richard._.Milhouse._.Nixon", "Madonna")
halveString(str2, pattern = encodeString("_."), first=FALSE)
```

insert.col

Insert Columns into a data.frame at a Specific Position

Description

Insert columns into a data.frame at a specific position. Transforms tibble or data.table to data.frame.

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", minRow = 1, onlyWarn=TRUE, verbose=TRUE)
```

Arguments

dat	An object which is intended to be a data.frame.
name	Optional: name of data.frame for use in messages
minRow	When used internally, function report when data.frame has less rows than specified in minRow.
onlyWarn	If TRUE, function warns if data.frame has less rows than specified in minRow. Otherwise, functions aborts with an error message.
verbose	Logical: print messages to console?

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)
```

makeTria	<i>Reshapes an unordered covariance/correlation matrix into triangular shape</i>
----------	--

Description

Function is mainly used for `eatAnalysis::wtdHetcor` function from the `eatAnalysis` package (<https://github.com/beckerbenj/eatAnalysis/>) and the `eatModel::q3FromRes` function in the `eatModel` package: Triangular covariance/correlation matrices are tidily reshaped.

Usage

```
makeTria (dfr)
```

Arguments

`dfr` A data frame consisting of a row name column and a square matrix.

Details

covariance/correlation matrices which are inherently symmetrical are often displayed in a space-saving manner by only showing the upper or lower triangular part, omitting the symmetrical counterpart. In R, covariance/correlation matrices tend to be displayed with their upper and lower halves. Whereas `lower.tri` and `upper.tri` allows to replace upper or lower half with NAs, the triangular shape could then be lost if the covariance/correlation matrix was provided in a long format and reshaped afterwards. `makeTria` sorts rows and columns appropriately to provide triangular shape if redundant entries are replaced by NA. Please note that the functions expects row names in the first column of the input data.frame.

Value

data frame.

Examples

```
dfr <- data.frame ( vars = paste0("var", 2:4), matrix(c(1:3, NA, NA, 5, 4,NA,6),
  nrow=3, ncol=3, dimnames=list(NULL, paste0("var", 1:3))))
makeTria(dfr)
```

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 (combinations 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", "common", "convert"))
```

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?
<code>suffixes</code>	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).
<code>setAttr</code>	Logical: restore the variable attributes? If <code>FALSE</code> , the behavior of <code>mergeAttr</code> equals the behavior of <code>merge</code> .

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"), or messages concerning additional common variables (except by-variables; "common"), or messages concerning converting of tibbles, tbls to data.frames ("convert"). 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"))

### In the following, the original examples of the `merge` function from the base
### package are called again using `mergeAttr` to illustrate the messages output in each case.

authors <- data.frame(
  surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4)))
authorN <- within(authors, { name <- surname; rm(surname) })
books <- data.frame(
  name = c("Tukey", "Venables", "Tierney",
           "Ripley", "Ripley", "McNeil", "R Core"),
  title = c("Exploratory Data Analysis",
            "Modern Applied Statistics ...",
            "LISP-STAT",
            "Spatial Statistics", "Stochastic Simulation",
            "Interactive Data Analysis",
            "An Introduction to R"),
  other.author = c(NA, "Ripley", NA, NA, NA, NA,
                  "Venables & Smith"))

(m0 <- mergeAttr(authorN, books, setAttr=FALSE))
(m1 <- mergeAttr(authors, books, by.x = "surname", by.y = "name", setAttr=FALSE))
m2 <- mergeAttr(books, authors, by.x = "name", by.y = "surname", setAttr=FALSE)
stopifnot(exprs = {
  all.equal(m0, m2[, names(m0)], check.attributes = FALSE)
  as.character(m1[, 1]) == as.character(m2[, 1])
  all.equal(m1[, -1], m2[, -1][ names(m1)[-1] ])
})

## "R core" is missing from authors and appears only here :
mergeAttr(authors, books, by.x = "surname", by.y = "name", all = TRUE, setAttr=FALSE)

## example of using 'incomparables'
x <- data.frame(k1 = c(NA,NA,3,4,5), k2 = c(1,NA,NA,4,5), data = 1:5)
y <- data.frame(k1 = c(NA,2,NA,4,5), k2 = c(NA,NA,3,4,5), data = 1:5)
mergeAttr(x, y, by = c("k1","k2"), setAttr=FALSE) # NA's match
mergeAttr(x, y, by = "k1", setAttr=FALSE) # NA's match, so 6 rows

```

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	<i>Drop rows containing missing values</i>
-------------------	--

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

*Transform continuous variables into ordered factors***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	<i>Combine data.frames by row, using only common columns.</i>
--------------	---

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)
```

rbind_fill_vector	<i>Combine vectors of unequal length by row, filling missing columns with NA.</i>
-------------------	---

Description

rbinds a list of vectors of unequal length to a data.frame. Missing columns are filled with NA.

Usage

```
rbind_fill_vector(x)
```

Arguments

x A list of vectors. Each element of x must have a dimension of NULL.

Value

a single data frame

Examples

```
a <- list(NULL, 1:2, NA, "a", 11:13)
rbind_fill_vector(a)
```

readMultisep

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

Description

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

Usage

```
readMultisep(file, sep, colnames=TRUE)
```

Arguments

file the name of the file which the data are to be read from.
sep the field separator character(s).
colnames logical. Whether first line in file contains colnames.

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

<code>var</code>	a vector (e.g. numeric, character, or factor)
<code>lookup</code>	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	<i>Removes all non-numeric characters from a string.</i>
------------------	--

Description

Function removes all non-numeric characters from a string.

Usage

```
removeNonNumeric ( string)
```

Arguments

<code>string</code>	a character vector
---------------------	--------------------

Value

a character string

Author(s)

Sebastian Weirich

Examples

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

removeNumeric

Removes alphanumeric characters from a string.

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(".dl.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 of length 1

Value

a character string

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

dat	A data.frame.
digits	Integer indicating the number of decimal places.

Value

Returns the rounded data.frame.

Examples

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

seq2

*Sequence generation***Description**

Creates a sequence of integers. Modified version of seq returning an empty vector if the starting point is larger than the end point. Originally provided by rlang::seq2().

Usage

```
seq2(from, to)
```

Arguments

from	The starting value of the sequence. Of length 1.
to	The end value of the sequence. Of length 1.

Value

A numerical sequence

Examples

```
seq2(from = 1, to = 5)
```

set.col.type

*Set the Class of Columns in a Data Frame***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

dat	A data frame
col.type	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)
verbose	if TRUE details about converted columns are printed on the console
...	Additional arguments to be passed to asNumericIfPossible

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

Creates skeleton for frequency tables with desired values

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

Frequency table for data frames, e.g. across multiple columns

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 table for more details.

Value

A frequency table

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

<code>datWide</code>	Data set in the wide format, i.e. one row per person
<code>noImp</code>	character vector of non-imputed variables which are desired for following analyses
<code>imp</code>	Named list of character vectors which include the imputed variables which are desired for following analyses
<code>multipleColumns</code>	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 <code>melt</code> function from the <code>reshape2</code> package).
<code>variable.name</code>	Applies only if <code>multipleColumns = "FALSE"</code> : name of variable used to store measured variable names
<code>value.name</code>	Applies only if <code>multipleColumns = "FALSE"</code> : 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	<i>Computed weighted frequency tables</i>
----------	---

Description

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

Usage

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

Arguments

<code>x</code>	a character or category or factor 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 frequency table

Examples

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

wtdVar	<i>Computed weighted variance</i>
--------	-----------------------------------

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)
```

%%%

Extract Parts of an Object (list)

Description

%%% is an operator that is mainly used internally in the eatRep and eatModel packages. %%% is similar to \$, but gives error instead of NULL if the corresponding element does not exists.

Usage

```
x %%% y
```

Arguments

x	a list
y	name of the corresponding element of x

Value

the selected element of the list x

Examples

```
## Not run:
x <- list(value1 = 14, value2 = NULL)
x$value2           # NULL
x$value_not_defined # NULL
x%%$value2         # NULL
x%%$value_not_defined # error

## End(Not run)
```

Index

%%\$, 36

addLeadingZerosToCharInt, 3

asNumericIfPossible, 3, 4, 6

cat, 23

catch_asNumericIfPossible, 3, 5

checkBackgroundVariables, 6

cleanifyString, 8

contr.wec.weighted, 9

crop, 10

data.timss1, 10

descr, 3, 11

do_call_rbind_withName, 12

eatTools-package, 3

existsBackgroundVariables, 7, 13

facToChar, 3, 14

gsubAll, 3, 15

halveString, 3, 15

insert.col, 16

lower.tri, 18

makeDataFrame, 17

makeTria, 18

merge, 19, 20

mergeAttr, 19

message, 23

multiseq, 21

na_omit_selection, 22

num.to.cat, 23

print_and_capture, 23

pwc, 24

rbind_common, 25

rbind_fill_vector, 25

read.spss, 19

readMultisep, 26

recodeLookup, 27

removeNonNumeric, 3, 27

removeNumeric, 3, 28

removePattern, 3, 29

roundDF, 29

seq2, 30

set.col.type, 3, 30

stop, 23

tablePattern, 31

tableUnlist, 32

upper.tri, 18

whereAre, 33

wideToLong, 33

wtdTable, 3, 35

wtdVar, 3, 35