Package ‘eatTools’

October 20, 2014

Type Package

Title eatTools

Version 0.0.12

Depends R (>= 2.14.0)

Imports psych, xlsx, car, plyr

Date 2014-10-17

Author Nicole Haag, Karoline Sachse, Sebastian Weirich, Martin Hecht and Thilo Siegle

Maintainer eat authors <eat-commits@lists.r-forge.r-project.org>

Description A collection of tools for the eat packages

License GPL (>= 2)

LazyLoad yes

LazyData yes

Repository R-Forge

Repository/R-Forge/Project eat

Repository/R-Forge/Revision 438

Repository/R-Forge/DateTimeStamp 2014-10-17 21:40:47

Date/Publication 2014-10-17 21:40:47

R topics documented:

asNumericIfPossible *Convert Columns of a Data Frame Into Numeric Values If Possible*

Description

This function converts 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. Optionally, only a logical vector indicating which columns are convertible is returned.

Usage

```
asNumericIfPossible(dat, set.numeric = TRUE, transform.factors = FALSE,
                    maintain.factor.scores = TRUE, verbose = TRUE)
```

Arguments

`dat` A data frame which should be converted.

`set.numeric` Logical: If `TRUE`, a data frame with all convertible columns converted to class `numeric` is returned. If `FALSE`, a logical vector indicating which columns are convertible to class `numeric`.

`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.

`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`.

`verbose` Logical: If `TRUE`, information about the class of the columns in the data.frame is given on the console.

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.

Value

Either a logical vector indicating which columns in the data frame are convertible to class `numeric` according to the specified conditions or a data frame in which all convertible columns are converted to class `numeric`.

Author(s)

Sebastian Weirich

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)
```

`collapseMissings` *Recode Character Missings of Different Types to 0 or* `NA`

Description

This function is used to recode character missings in datasets that were prepared with functions from the `eatPrep` package to 0 or `NA`. It is called by several functions of the `eat` package family.

Usage

```
collapseMissings(dat, missing.rule = NULL, items)
```

Arguments

`dat` data frame containing character missings (e.g. type `mbd` - missing by design)

`missing.rule` A list with definitions how to recode the different types of missings in the dataset. If `NULL`, the default described in 'Details' is used.

`items` A character vector containing the column names of the data frame for which character missings are to be recoded.

Details

One of the main ideas of the `eat` package family is that different types of missing values should remain distinguishable during data preparation, thus allowing the user to flexibly recode them to different values during the IRT scaling process. `collapseMissings` can be used to facilitate the recoding of the different types of character missings before scaling or when exporting the data to other software packages (e. g., SPSS).

The `eat` package family currently supports six different types of missings, namely mvi (text volume insufficient): used in writing tasks if a person wrote to little to evaluate whether they met a specific criterion.

mnr (missing not reached): used whenever a person did not reach the respective task in his or her test booklet. All consecutive missing values clustered at the end of a test session can be coded mnr, e.g., by the function recodeMbiToMnr from package eatPrep.

mci (missing coding impossible): used whenever a response cannot be coded due to technical problems (e.g., problems in digitalizing the booklets)

mbd (missing by desing): used whenever an item was not administered to a specific person.

mir (missing invalid response): used whenever a person attempted to answer an item but this answer cannot be classified in the existing coding scheme. Can also be used for multiple choice-items when the respondent selected more than one option.

mbi (missing by intention): used whenever a person was expected to answer an item but did not provide a response.

The default recode values for these missing types are: text volume insufficient = 0, missing not reached = 0, missing coding impossible = NA, missing by design = NA, missing invalid response = 0, missing by intention = 0

Value

A data frame with recoded missings.

Author(s)

Karoline Sachse, Martin Hecht

References

OECD (2005). *PISA 2003 Technical Report*. OECD Publishing.

commonItems *Identify Common Items for Several Groups*

Description

This function identifies sets of items that have been administered to two groups of persons.

Usage

```
commonItems(dat, group.var, na = NA, uncommon = FALSE, simplify = TRUE)
```

Arguments

dat	A data frame with item responses and a grouping variable.
group.var	Name or column number of the group variable in dat
na	A character string indicating which value should be considered as not administered (missing by design)
uncommon	if TRUE a vector of items that have only been administered to one of the two groups is additionally returned.
simplify	if TRUE a character vector is returned (only in case of 2 groups and uncommon=FALSE)

Details

dat must only contain the group variable and the items, if further variables are in dat they are treated as items. If group.var specifies more than two groups, pairwise group comparisons are performed.

Value

returns a list of all group.var combinations with character vectors of common item names. If uncommon=TRUE a vector of uncommon (unique) items of each group is additionally returned.

The names of list elements are the two group names concatenated by "|".

Author(s)

Martin Hecht

See Also

commonItems.percent

Examples

```
data(science1)
d <- science1[ , c("version", science1.items)]

# common items are listed for each combination of groups
str(commonItems(dat = d, group.var = "version", na = "mbd"))

# uncommon items are returned as well
str(commonItems(dat = d, group.var = "version", na = "mbd", uncommon = TRUE))
```

---

commonItems.percent *Identify Percentage of Common Items for Several Groups*

Description

This function calculates the percentage of items that have been administered to two groups of persons.

Usage

```
commonItems.percent(dat, group.var, na = NA, xlsx = NULL)
```

Arguments

dat	A data frame with item responses and a grouping variable.
group.var	Name or column number of the group variable in dat
na	A character string indicating which value should be considered as not administered (missing by design)
xlsx	Optional: Full path of Excel file for results.

Details

dat must only contain the group variable and the items, if further variables are in dat they are treated as items. If group.var specifies more than two groups, pairwise group comparisons are performed.

Value

returns a data frame with common item percentage(s)

Author(s)

Martin Hecht

See Also

commonItems

Examples

```
data(science1)
d <- science1[ , c("version", science1.items)]
commonItems.percent(dat = d, group.var = "version", na = "mbd")
```

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

fill.na *Replace Missing Values in a Vector*

Description

Missing values in a vector are replaced by the last (forward) or next (backward) observed value.

Usage

```
fill.na(vec, backwards = FALSE, na.rm = FALSE)
```

Arguments

vec	a vector
backwards	if FALSE NAs are replaced by the last observed value, if TRUE NAs are replaced by the next observed value
na.rm	if TRUE NAs at the start and end of vector are removed

Details

In the clinical literature, the procedure of replacing a missing value with the last observed value is known as the "Last Observation Carried Forward" imputation technique. However, there is a large body of literature suggesting that this method may lead to biased estimates of means and covariances and should therefore be avoided for imputation.

Value

A vector with replaced missing values.

Author(s)

Martin Hecht

Examples

```
(vec <- c(NA, 1, NA, NA, 2, NA, 3, NA))
fill.na(vec)
fill.na(vec, backwards = TRUE)
```

make.dummies *Create Dummy Variables from a Data Frame*

Description

Create dummy variables using dummy.code from the psych package. The dummy variables' names can be customized and the variables can be added to the input data frame.

Usage

```
make.dummies(dat, cols, colname.as.prefix = TRUE, delimiter = ".",
capitalize = FALSE, nchar = NULL, add = TRUE, sort.into.dat = TRUE,
oneToColname = FALSE, zeroToNA = FALSE, factor.indices = FALSE )
```

Arguments

`dat`	A data frame
`cols`	colnames of variables to be dummy coded
`colname.as.prefix`	Logical: If `TRUE` the original variable name is added as prefix
`delimiter`	A character string by which the variable name and the level name will be separated (only evaluated if `colname.as.prefix = TRUE`)
`capitalize`	Logical: If `TRUE` the level names are capitalized
`nchar`	Number of characters the level names should be truncated to
`add`	Logical: If `TRUE` the dummy variables are appended to `dat`
`sort.into.dat`	Logical: If `TRUE` (and `add = TRUE`) the dummy variables are added and sorted into `dat` according to their column names
`oneToColname`	Logical: If `TRUE`, the values of cases with a value of 1 on the dummy variable are set to the colname of respective column. This changes the column class of the dummy variable(s) from `numeric` to `character`.
`zeroToNA`	Logical: If `TRUE`, the values of cases with a value of 0 on the dummy variable are set to `NA`.
`factor.indices`	Logical: If `TRUE`, numeric indices of factor levels are used instead of factor level names.

Value

A data frame with dummy variables. Depending on `add` the returned object contains either the original data frame with the dummy variables appended or only the dummy variables.

Author(s)

Martin Hecht

Examples

```
## Not run:
data(science1)

science1.dum <- make.dummies(science1, c("sex","booklet"))
str(science1.dum[,1:12])

science1.dum <- make.dummies(science1, c("sex","booklet"), nchar = 1)
str(science1.dum[,1:12])

science1.dum <- make.dummies(science1, c("sex","booklet"), delimiter = "_")
str(science1.dum[,1:12])

science1.dum <- make.dummies(science1, c("sex","booklet"), delimiter = "", capitalize = TRUE)
str(science1.dum[,1:12])
```

```
science1.dum <- make.dummies(science1, c("sex","booklet"), colname.as.prefix = FALSE)
str(science1.dum[,1:12])

science1.dum <- make.dummies(science1, c("sex","booklet"), sort.into.dat = FALSE)
str(science1.dum[ , (ncol(science1.dum)-9):ncol(science1.dum)])

science1.dum <- make.dummies(science1, c("sex","booklet"), add = FALSE)
str(science1.dum)

science1.dum <- make.dummies(science1, c("sex","booklet"), oneToColname = TRUE, zeroToNA = TRUE)
str(science1.dum[,1:12])

science1.dum <- make.dummies(science1, c("sex","booklet"), factor.indices = TRUE)
str(science1.dum[,1:12])


## End(Not run)
```

modus *Compute the Mode of a Variable*

Description

Calculate the mode (most frequent value) of a variable

Usage

```
modus(x, randTies = FALSE)
```

Arguments

x	a vector
randTies	If TRUE, in case more than one mode is found, one random value of all modes is returned.

Details

The modus function is designed to always return only one value for the mode of a variable. If the variable is bimodal or multimodal, the function returns either NA (if randTies = FALSE) or a randomly chosen value of all modes (if randTies = TRUE).

Value

the mode (most frequent value) of the variable

Author(s)

Martin Hecht

Examples

```
## Not run:
x <- c(1, 1, 2, 2)
modus(x)
modus(x, randTies = TRUE)

x <- c(1, NA, NA)
modus(x)

x <- c("x", "x", "y")
modus(x)

## End(Not run)
```

`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)
```

`reading_writing` *Reading and writing achievement test data*

Description

This data set contains fictional achievemetn scores of 4619 students of 3 countries. See format for a short description of variables.

Usage

```
data(reading_writing)
```

Format

'data.frame': 4619 obs. of 24 variables

idstud Unique identifier

wgtSTUD variable of individual student weights

sex Examinee's sex

country Country where the examinee is from.

JKZone jackknifing zone

JKrep replicate ID

reading_score1 First plausible value of the reading score

reading_score2 Second plausible value of the reading score

reading_score3 Third plausible value of the reading score

writing_score1 First plausible value of the writing score

writing_score2 Second plausible value of the writing score

writing_score3 Third plausible value of the writing score

passed_reading1 First indicator whether examinee passed the reading minimal requirement

passed_reading2 Second indicator whether examinee passed the reading minimal requirement

passed_reading3 Third indicator whether examinee passed the reading minimal requirement

passed_writing1 First indicator whether examinee passed the writing minimal requirement

passed_writing2 Second indicator whether examinee passed the writing minimal requirement

passed_writing3 Third indicator whether examinee passed the writing minimal requirement

zehisei Overall five indicators of highest socio-economic status. Think of it as imputed values.

income Overall two indicators of mean month income.

Source

Simulated data

reinsort.col *Insert Columns into a Data Frame in a Specific Position*

Description

Insert columns into a data frame in specific position

Usage

```
reinsort.col(dat, toreinsort, after)
```

Arguments

dat	A data frame
toreinsort	Column name(s) or column number(s) of the colums to be reinserted
after	Column name or column number after which the colums specified in reinsort should be reinserted.

Value

A data frame with columns in specified positions.

Author(s)

Martin Hecht

rmNA *Remove Columns and Rows with Missing Data*

Description

Remove columns and rows that contain only missing values from a data frame or a matrix

Usage

```
rmNA(dat, remove = TRUE, verbose = FALSE)
```

Arguments

dat	A data frame or a matrix
remove	if TRUE columns and rows containing only missing values are removed, if FALSE a list of identified columns and rows is returned
verbose	if TRUE the removed columns and rows are printed on the console.

Value

Either a list indicating which columns and which rows in the data contain only missing values or a data frame or matrix in which all columns and rows containing only missing values are removed.

Author(s)

Martin Hecht

See Also

rmNAcols, rmNArows

Examples

```
(mat <- matrix(c(1,1,1,1,1,NA, 1,1,1,1,NA,NA, 1,1,1,NA,NA,NA, 1,1,NA,NA,NA,NA,
               1,NA,NA,NA,NA,NA, NA,NA,NA,NA,NA,NA), ncol=6, byrow=TRUE))
rmNA(mat, verbose = TRUE)
rmNA(mat, remove = FALSE)
```

rmNAcols *Remove Columns with Missing Data*

Description

Remove columns containing missing values from a data frame or a matrix

Usage

```
rmNAcols(dat, rows = NULL, tolerance = 0, cumulate = TRUE,
             remove = TRUE, verbose = FALSE)
```

Arguments

dat	A data frame or a matrix
rows	rows to include in evaluating the missing values of columns, can be a list of vectors to specify row subsets
tolerance	Number of non-NA cells that are "tolerated", can be a list corresponding to rows
cumulate	if TRUE, tolerance is cumulated; if FALSE, exact tolerance is used
remove	if TRUE columns and rows are removed, if FALSE identified columns are returned
verbose	if TRUE the removed columns are printed on the console

Value

depends on option remove

Author(s)

Martin Hecht

See Also

calls rmNA and rmNArows

Examples

```
# example matrix
(mat <- matrix(c(1,1,1,1,1,1, 1,1,1,1,1,NA, 1,1,1,1,NA,NA, 1,1,1,NA,NA,NA,
1,1,NA,NA,NA,NA, 1,NA,NA,NA,NA,NA, NA,NA,NA,NA,NA,NA), ncol=7))

# remove column with entirely NA (column 7)
rmNAcols(mat, verbose = TRUE)

# remove column with NA on rows 3, 4, 5 (columns 5, 6, 7)
rmNAcols(mat, c(3,4,5), verbose = TRUE)
rmNAcols(mat, c(-1,-2,-6), verbose = TRUE)

# tolerance=1 , 1 non-NA is permitted (columns 6 and 7)
rmNAcols(mat, tolerance=1, verbose = TRUE)

# tolerance=6 , 6 non-NA are permitted (all columns are removed)
rmNAcols(mat, tolerance=6, verbose = TRUE)

# do not cumulate / exact tolerance (column 1)
rmNAcols(mat, tolerance=6, cumulate=FALSE, verbose = TRUE)

# two subsets of rows
rmNAcols(mat, rows = list(c(1, 2), c(4, 5)), verbose = TRUE)

# two subsets of rows with different tolerance
rmNAcols(mat, rows = list( 1, c(2, 3, 4, 5)), tolerance = list(0, 1), verbose = TRUE)

# identify cols, no deletion
rmNAcols(mat, rows = list(c(1, 2), c(3, 4, 5)), tolerance = list(0, 1), remove = FALSE)
```

---

rmNArows *remove NA rows from data*

Description

remove rows that are completely or partially NA from data.frame or matrix

Usage

```
rmNArows(dat, cols = NULL, tolerance = 0, cumulate = TRUE,
         remove = TRUE, verbose = FALSE)
```

Arguments

`dat`	data.frame or matrix
`cols`	columns to include, can be a list of vectors to specify column subsets
`tolerance`	number of non-NA cells that are "tolerated", can be a list corresponding to `cols`
`cumulate`	if `TRUE`, `tolerance` is cumulated; if `FALSE`, exact `tolerance` is used
`remove`	if `TRUE`, columns and rows are removed; if `FALSE`, identified rows are returned
`verbose`	if `TRUE` removed columns and rows are printed on output window

Value

depends on option `remove`

Author(s)

Martin Hecht

See Also

calls `rmNA` and `rmNAcols`

Examples

```
# example matrix
(mat <- matrix(c( 1,1,1,1,1, 1,1,1,1,NA, 1,1,1,NA,NA, 1,1,NA,NA,NA,
  1,NA,NA,NA,NA, NA,NA,NA,NA,NA), ncol=5, byrow=TRUE))

# remove row with entirely NA (row 6)
rmNArows(mat, verbose = TRUE)

# remove row with NA on column 3, 4, 5 (rows 4, 5, 6)
rmNArows(mat, c(3,4,5), verbose = TRUE)
rmNArows(mat, c(-1,-2), verbose = TRUE)

# tolerance=1 , 1 non-NA is permitted (rows 5 and 6)
rmNArows(mat, tolerance=1, verbose = TRUE)

# tolerance=5 , 5 non-NA are permitted (all rows are removed)
rmNArows(mat, tolerance=5, verbose = TRUE)

# do not cumulate / exact tolerance (row 1 is removed)
rmNArows(mat, tolerance=5, cumulate=FALSE, verbose = TRUE)
rmNArows(mat, tolerance=5, cumulate=FALSE, remove = FALSE)

# two subsets of columns
rmNArows(mat, cols = list(c(1, 2), c(4, 5)), verbose = TRUE)

# two subsets of columns with different tolerance
rmNArows(mat, cols = list(c(1), c(2, 3, 4, 5)), tolerance = list(0, 1), verbose = TRUE)

# identify rows, no deletion
rmNArows(mat, cols = list(c(1), c(2, 3, 4, 5)), tolerance = list(0, 1), remove = FALSE)
```

`science1` *Science achievement test data*

Description

This data set contains responses of 420 students on 185 science items. Additional variables are included: id, grade, sex, booklet, track, version, and four dummy coded variables that indicate Track x Version groups. An incomplete block design was used with 4 booklets. Codes on items are: "0" - wrong "1" - right "mbd" - missing by design "mbi" - missing by intention "mir" - missing due to irregular response

Usage

```
data(science1)
```

Format

'data.frame': 420 obs. of 195 variables

Source

Simulated data

science1.context.vars *Science achievement test data - Context variable names*

Description

This vector contains the names of context variables in data set `science1`

Format

chr [1:9]

science1.item.characteristics *Science achievement test data - Item characteristics*

Description

This data frame contains item characteristics for usage with `automateModels` and data set `science1`

Format

'data.frame': 185 obs. of 3 variables

science1.items *Science achievement test data - Item names*

Description

This vector contains the names items in data set `science1`

Format

chr [1:185]

science1.scales *Science achievement test data - Scale definition*

Description

This data frame contains scale definitions for usage with automateModels and data set science1

Format

'data.frame': 185 obs. of 7 variables

science1.testlets *Science achievement test data - Testlet definition as dummy codes*

Description

This data frame contains testlet definitions as dummy codes for usage with automateModels and data set science1

Format

'data.frame': 185 obs. of 54 variables

set.col.type *Set the Class of Columns in a Data Frame*

Description

Convert the Class 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

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))
```

Index