Package 'eat'

February 23, 2012

Type Package

Title eat

Version 1.4.8

Depends R(>= 2.14.0), sendmailR, gdata, xlsx, car, reshape, foreign,date, plyr, parallel

Date 2012-02-23

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

Maintainer Martin Hecht <martin.hecht@iqb.hu-berlin.de>

Description The package eat is designed to simplify data preparation and IRT modeling with the software ConQuest within the R programming environment. It includes routines for automation of data preprocessing and an interface to specify and run several IRT models.

License GPL

LazyLoad yes

LazyData yes

R topics documented:

`eat-package` *eat-package*

Description

More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package:	eat
Type:	Package
Version:	
Date:	
License:	What license is it under?
LazyLoad:	yes

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Who wrote it

Maintainer: Who to complain to <yourfault@somewhere.net> ~~ The author and/or maintainer of the package ~~

References

~~ Literature or other references for background information ~~

`aggregateData` *Aggregate Datasets with Missing Values*

Description

Aggregates datasets with constraints on missing values

Usage

```
aggregateData(dat, subunits, units, aggregatemissings = "use.default", rename = FALSE, recodedDa
```

Arguments

`dat`	A data frame.
`subunits`	A data frame with subunit information. See 'Details'.
`units`	A data frame with unit information. See 'Details'.
`aggregatemissings`	Either the character string `"use.default"` or a *n x n* matrix with information on how missing values should be aggregated. See 'Details'.
`rename`	Should units with only one subunit be renamed to their unit name? Default is `FALSE`.
`recodedData`	Logical indicating whether colnames in dataset to aggregate are the subunit names (as in `subunits$subunit`) or recoded subunit names (as in `subunits$subunitRecoded`). Default is `TRUE`, meaning that colnames are recoded subitem names.

Details

`aggregateData` aggregates units in data frames with special consideration of missing values. The aggregation of missing values is specified in argument `aggregatemissings`.

The results of `aggregateData` will be written to a protocol file with `sunk`.

Examples of data frames `subunits` and `units` can be found via `data(inputList)`.

Value

A data frame with aggregated units and, if `rename = TRUE`, renamed subunits.

Warning

Missings are only correctly aggregated if their values correspond to the values given in `aggregatemissings`. `aggregateData` does not check for value types or whether codes are valid. Use of `checkData` and `recodeData` before using `aggregateData` is therefore strongly recommended.

Author(s)

Nicole Haag, Anna Lenski

References

For missing types see http://code.google.com/p/zkdlib/wiki/MissingHandling

See Also

`recodeData`, `checkData`

Examples

```
data(inputDat)
data(inputList)

dat1 <- inputDat[[1]]  # get first dataset from inputDat
datRec <- recodeData(dat1, inputList$values, inputList$subunits) # recode Data first
datAggr <- aggregateData(datRec, inputList$subunits, inputList$units, rename = TRUE, recodedData = TRUE)
```

---

asNumericIfPossible *Transform columns of a data.frame into numeric values if possible*

Description

In contrast to `as.numeric`, Function transforms only "transformable" columns of a data.frame into numeric values (i.e. without creating `NA` when transformation fails. Non-transformable columns are maintained. Optionally, only a logical vector is given, indicating which columns are transformable.

Usage

```
asNumericIfPossible ( dataFrame, set.numeric = TRUE, transform.factors = FALSE, maintain.factor.
```

Arguments

`dataFrame`	A data.frame which columns should be transformed.
`set.numeric`	Logical: If `TRUE`, data.frame with transformed columns is returned. If `FALSE`, a logical vector is returned, indicating which columns are transformable.
`transform.factors`	Logical: Should columns of class `factor` transformed? If `FALSE`, columns of class `factor` are maintained. If `TRUE`, columns of class `factor` are attempted to transform.
`maintain.factor.scores`	Logical. Only relevant if `transform.factors = TRUE`. If `TRUE`, the nominal values of the factor are transformed if possible. If `FALSE`, the integer numbers representing the factors' nominal values are returned. See details.

verbose Logical: If TRUE, informations about the class of the columns in the data.frame are printed to the console.

Details

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

Value

Either a logic vector, indicating which columns in the data.frame are transformable according to the specified conditions, ora data.frame in which transformable columns are transformed.

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

---

automateConquestModel *automateConquestModel*

Description

Wrapper function which calls several functions to build and write Conquest input files.

Usage

```
automateConquestModel(dataset, ID, regression=NULL, DIF=NULL, group.var=NULL,
weight=NULL, testitems, na=list(items=NULL, DIF=NULL, HG=NULL, group=NULL,
weight=NULL), person.grouping=NULL, item.grouping=NULL, model.statement="item",
m.model="1pl", Title = NULL, jobName, jobFolder, subFolder=list(), dataName=NULL,
anchor=NULL, pathConquest, method=NULL,std.err=NULL,distribution=NULL,
n.plausible=NULL, set.constraints=NULL, nodes=NULL, p.nodes=NULL, f.nodes=NULL,
n.iterations=NULL, converge=NULL, deviancechange=NULL, name.unidim=NULL,
equivalence.table="wle",use.letters=FALSE, checkLink=FALSE, verbose = TRUE)
```

Arguments

dataset	data.frame containing all variables necessary for analysis
ID	name or column number of 'id' variable
regression	character vector with names or integer vector with column numbers of one or more 'context' variables (e.g. sex, school , ...)
DIF	character string with name or scalar with column number of only one variable denominating groups for which analysis of differential item functioning is to be done (e.g. sex, class , ...)
group.var	character vector with names or integer vector with column numbers of one or more 'group' variables (e.g. sex, school , ...)
weight	character string with name or scalar with column number of only one 'weighting' variable
testitems	character vector with names or integer vector with column numbers of 'item' variables (e.g. sex, school , ...)
na	List of numerical vector including numbers to be considered as 'sysmis'. Specific missing codes can be defined for each type of variable, e.g. testitems, DIF variables, ...
item.grouping	data.frame with grouping information of items, first column must be 'item' which includes item names, further columns contain scale definitions, 0 indicates that the respective item is NOT part of the scale, 1 indicates that this item is part of the scale, colnames of columns are the names of the scales
person.grouping	data.frame with grouping information of persons, first column must be the name of 'id' (e.g. idstud), further columns contain group definitions, 0 indicates that the respective person is NOT part of the group, 1 indicates that this person is part of the group, colnames of columns are the names of the groups
model.statement	character string which appears in Conquest Syntax as model statement. Set to `item` by default. When DIF variable is specified, statement is set to `item - [name of DIF variable] + item*[name of DIF variable]` by default. However, user's specification of `model.statement` overwrites default in each case.
m.model	character string specifying the IRT model used for analysis. At the time, only `"1PL"` is available.
Title	optional: character string with title of analysis which appears in Conquest Syntax. If no title is specified, informations about computer and user name and R version are printed in Conquest Syntax.
jobName	character string specifying name of analysis. All Conquest input and output files will named by `jobName` and their corresponding suffixes.
jobFolder	character string specifying the folder where all analysis files will be generated, for example `"C:/programme/analysis"`
subFolder	optional: List of character strings specifying maximal 2 optional subfolders relative to `jobFolder` for the data and the output files. Character strings must be named with `data` and `out`, for example `subFolder=list(data="../../dataset/analysis1", out="../../output/analysis1")` Double dots `..` indicates to move one level above in folder structure. For example, if `jobFolder` is `"C:/programme/analysis"` and `subFolder` is `list(data="../dataset/analysis1", out="../../output/analysis1")`,

dataset is written to "C:/programme/dataset/analysis1" and output is written to "C:/output/analysis1". When subFolder$data == NULL, dataset is written to the folder specified by jobFolder. Same is true for subFolder$out == NULL.

dataName — optional: character string specifying name of dataset if intend to differ from name specified by jobName. When dataName == NULL, dataset is named [jobName].dat

anchor — optional: data frame with anchor parameters. First column of anchor refers to item name, second columns refers to parameter used as reference for anchoring. Note: not all items in the data.frame have to occur as anchor parameters in the anchor data frame, as well as not all items in the anchor frame have to occur in the data frame.

pathConquest — character string with path and name of Conquest console, for example "c:/programme/conquest/co

method — optional: character string with method for analysis. Possible options are "gauss" (default), "quadrature", "montecarlo". See Conquest manual, pp.225.

std.err — optional: character string specifying types of standard errors to be estimated. Possible options are "full", "quick" (default), "none". See Conquest manual, pp.167.

distribution — optional: character string with a priori trait distribution. Possible options are "normal" (default), "discrete". See Conquest manual, pp.167.

n.plausible — optional: integer scalar specifying numbers of plausible values to draw. Default is 5.

set.constraints — optional: character string specifying the constraints of the scale. Possible options are "cases" (default), "items", "none". When anchor parameter are specified in anchor, constraints will set to "none" in each case.

nodes — optional: integer scalar specifying numbers of nodes in analysis. Default is 15.

p.nodes — optional: integer scalar specifying numbers of p nodes in analysis. Sets the number of nodes that are used in the approximation of the posterior distributions, which are used in the drawing of plausible values and in the calculation of EAP estimates. The default is 2000.

f.nodes — optional: integer scalar specifying numbers of f nodes in analysis. Sets the number of nodes that are used in the approximation of the posterior distributions in the calculation of fit statistics. The default is 2000.

n.iterations — optional integer scalar. Sets the maximum number of iterations for which estimation will proceed without improvement in the deviance. The minimum value permitted is 5. The default value is 20.

converge — optional scalar. Instructs estimation to terminate when the largest change in any parameter estimate between successive iterations of the EM algorithm is less than converge. The default value is 0.0001.

deviancechange — optional scalar. Instructs estimation to terminate when the change in the deviance between successive iterations of the EM algorithm is less than deviancechange. The default value is 0.0001.

name.unidim — optional: character string with name of one latent dimension, if not specified in labels.

equivalence.table — optional: character string specifying type of equivalence table to print. Possible options are "wle" (default), "mle" or NULL.

use.letters	logical: Should values coded als letters? May be relevant only in partial credit models comprising items with more than 10 categories to avoid columns with width 2 in Conquest.
checkLink	logical: If TRUE, items in dataset are checked for being connected with each other via design (function checkLink is called)
verbose	logical: If TRUE, messages are printed on console.

Value

No results are returned to console. Input files and batch string are written to disk in specified folder(s).

Author(s)

Sebastian Weirich, Karoline Sachse, Martin Hecht

---

automateDataPreparation

automateDataPreparation

Description

prepare datasets for automateModels

Usage

```
automateDataPreparation( inputDat = NULL, inputList, path = NULL, loadSav,
checkData, mergeData, recodeData,aggregateData, scoreData, writeSpss,
filedat = "zkddata.txt", filesps = "readZkdData.sps",
aggregatemissings = "use.default", rename = TRUE, recodedData = TRUE,
correctDigits=FALSE, truncateSpaceChar = TRUE, newID = NULL, oldIDs = NULL,
missing.rule = list(mvi=0, mnr=0, mci=0, mbd=NA, mir=0, mbi=0))
```

Arguments

inputDat	A list of data frames if no .sav files shall be read in.
inputList	A list of data frames containing additional information (see Details).
path	A character string containing the path where the logfolder will be created. Also required by loadSav (source of SPSS files) and writeSpss. Default is the current R working directory.
loadSav	logical (whether function loadSav shall be called).
checkData	logical (whether function checkData shall be called).
mergeData	logical (whether function mergeData shall be called).
recodeData	logical (whether function recodeData shall be called for subunits).
aggregateData	logical (whether function aggregateData shall be called).
scoreData	logical (whether function recodeData shall be called for units).
writeSpss	logical (whether function writeSpss shall be called).
filedat	A character string with the name of the output data file required by writeSpss.

filesps	A character string with the name of the output syntax file required by writeSpss.
missing.rule	A list containing recode information for character missings required by writeSpss. See 'References' for description of default values.
aggregatemissings	A character string. Either "use.default" or "seeInputList", if pattern was specified in inputList$aggrMiss.
rename	logical. See aggregateData.
recodedData	logical. See aggregateData.
correctDigits	logical. See loadSav.
truncateSpaceChar	logical. See loadSav.
newID	A character string containing the case IDs name in the final data frame. Default is "ID" or a character string specified in inputList sheet 6 (see readDaemonXlsx).
oldIDs	A vector of character strings containing the IDs names in the original datasets. Default is as specified in inputList$savFiles.

Details

inputList is a list of data frames. It can be created either by ZKDaemon via readDaemonXlsx or by makeInputLists. Compulsory: units, subunits, values. Optional: unitRecodings, savFiles, newID, aggregateMissings.

Value

A single data frame in last transformation status.

Author(s)

Karoline Sachse

References

http://code.google.com/p/zkdlib/wiki/MissingHandling

Examples

```
inpList <- inputList
inpDat <- inputDat
test <- automateDataPreparation (inputList=inpList, inputDat = inpDat,
        path = "c:/temp/test_eat", loadSav = FALSE, checkData=TRUE,
        mergeData = TRUE, recodeData=TRUE, aggregateData=TRUE, scoreData= TRUE,
        writeSpss=TRUE)
```

automateModels *automateModels*

Description

specify and run several ConQuest models

Usage

```
automateModels(dataset, id = NULL, context.vars = NULL, items = NULL,
item.grouping = NULL, select.item.group = NULL, person.grouping.vars = NULL,
person.grouping.vars.include.all = FALSE, person.grouping = NULL,
select.person.group = NULL, checkLink = FALSE, additional.item.props = NULL, folder,
overwrite.folder = TRUE, analyse.name.prefix = NULL, analyse.name = NULL,
analyse.name.elements = NULL, data.name = NULL, m.model = NULL, software = NULL,
dif = NULL, weight = NULL, anchor = NULL, regression = NULL,
adjust.for.regression = FALSE, q3 = FALSE, missing.rule = NULL, cross = NULL,
subfolder.order = NULL, subfolder.mode = NULL, allNAdelete = TRUE, additionalSubFolder = NULL,
run.mode = NULL, n.batches = NULL, run.timeout = 1440, run.status.refresh = 0.2,
all.local.cores = TRUE, email = NULL, smtpServer = NULL, write.txt.dataset = FALSE,
delete.folder.countdown = 5, conquestParameters = NULL )
```

Arguments

dataset	data.frame containing all variables type of variables ("id" , "context.vars" or "items") must be set using options id, context.vars, items
id	name or column number of 'id' variable in dataset
context.vars	names or column numbers of 'context' variables (e.g. sex, school , ...) in dataset
items	names or column numbers of 'item' variables in dataset if omitted, all variables that are not classified as 'id' or 'context' variables are treated as 'items'
item.grouping	data.frame with grouping information of items, first column must be 'item' which includes item names, further columns contain scale definitions, 0 indicates that the respective item is NOT part of the scale, 1 indicates that this item is part of the scale, colnames of columns are the names of the scales
select.item.group	character vector of scale names chosen for analysis
person.grouping.vars	character vector of 'context' variables in dataset which are used to automatically generate 'person.grouping', each category is transformed into the 'person.grouping' format
person.grouping.vars.include.all	logical vector (along person.grouping.vars), indicates whether to generate a variable 'all' for the specific variable
person.grouping	data.frame with grouping information of persons, first column must be the name of 'id' (e.g. idstud), further columns contain group definitions, 0 indicates that the respective person is NOT part of the group, 1 indicates that this person is part of the group, colnames of columns are the names of the groups

`select.person.group`
: character vector of group names chosen for analysis

`checkLink`
: logical: If `TRUE`, items in dataset are checked for being connected with each other via design (function `checkLink` is called) 23.02.2012: not yet implemented

`additional.item.props`
: data.frame of additional item information to be merged to model results, first column must be 'item' and contain item names

`folder`
: folder to write output into

`overwrite.folder`
: logical, if TRUE (default), folder is completely emptied

`analyse.name.prefix`
: prefix (e.g. "pilotStudy") to be attached to all analyses names

`analyse.name`
: analyses names are usually automatically set, if you want to set them manually use this option

`analyse.name.elements`
: analyses names are set automatically using these elements: c ("scale" , "group" , "dif" , "regression" , "anchor"), use this option to change composition and order of the analyses names generation

`data.name`
: optional: character string specifying name of dataset if intend to differ from name specified by jobName. When dataName == NULL, dataset is named [jobName].dat

`m.model`
: measurement model, "1pl" (default), "2pl", "3pl", "4pl"

`software`
: "conquest" (default) no other software implemented yet

`dif`
: variable that is used for differential item functioning

`weight`
: case weight variable

`anchor`
: data.frame with anchor information

`regression`
: variable(s) that is/are used

`adjust.for.regression`
: if TRUE item parameters (difficulty) are centered on the mean of the entire sample if FALSE (default) item parameters (difficulty) are centered on the mean of the regression reference group

`q3`
: Logical: If `TRUE`, Yen's Q3 statistic is computed.

`missing.rule`
: definition how to recode distinct missings in dataset

`cross`
: scales in 'item.grouping' and groups in 'person.grouping' can be crossed to define distinct analyses "all": scales and groups are crossed "item.groups", scales are separately (unidimensional) run (instead of one multidimensional model) "person.groups", person groups are separately (single group) run (instead of one multigroup model)

`subfolder.order`
: subfolders are automatically generated in this order c ("i.model" , "p.model" , "m.model" , "software" , "dif" , "regression" , "anchor")

`subfolder.mode`
: "none": no subfolders are created "full": complete subfolders are created according to 'subfolder.order' "intelligent" (default): meaningful subfolders are created

`allNAdelete`
: if `TRUE` all cases with complete missings on `items` are removed, if `FALSE` these cases are not deleted Note: this is a global option, that is set for all modelss

`additionalSubFolder`
: specification for 'data' and 'out' subfolder (constant over all analyses)

`run.mode`
: "serial": serial runs on local machine "parallel": batch files must be started manually (e.g. on several machines)

`n.batches`
: number of batch files that are created, batch files contain one or more analyses

`run.timeout`
: minutes to wait for analyses to finish, default: 1440 (24h)

`run.status.refresh`
: time for console refresh of model run status, default: 0.2 (12sec)

`all.local.cores`
: if `TRUE` and `run.mode="serial"` all cores of local machine are used for analysis

`email`
: set email address to receive an email when analyses are finished or time's up

`smtpServer`
: smtpServer for sending emails, default: "mailhost.cms.hu-berlin.de"

`write.txt.dataset`
: write out datasets as ascii, default: FALSE

`delete.folder.countdown`
: countdown for deletion of 'folder', default: 5 (seconds)

`conquestParameters`
: Set ConQuest parameters as a named list.
Available option are:
"pathConquest","method","std.err","distribution","n.plausible","set.constraints", "nodes","p.nodes","f.nodes","n.iterations","converge","deviancechange", "equivalence.table","use.letters","na","model.statement"
See automateConquestModel documentation for details.

Details

To run several models list parameters as corresponding lists Explicitly list NULL if parameter should not be set or be defaulted See examples

Value

returns results in specific format

Author(s)

Martin Hecht, Karoline Sachse, Sebastian Weirich, Christiane Penk, Malte Jansen, Sebastian Wurster

---

`bi.linking` *bi.linking*

Description

Links results from several analysis. Each analysis is linked with each other.

Usage

```
bi.linking ( results , scales=NULL , folder=NULL , file.name=NULL , method = NULL , lower.triang
```

Arguments

results	result list from automateModels run
scales	Character vector of scales for which linking should separately done. If NULL, all analysis in the results list are linked. Note: due to suboptimalities in development process, analysis name must contain scale!!
folder	output folder, will be emptied!
file.name	file.name for output excel, default: "bi.linking.results.xlsx"
method	set linking method to either "Mean-Mean" , "Haebara" or "Stocking-Lord" (default)
lower.triangle	set reference groups for the linking

Value

writes linking results to excel file. returns linking results as list.

Author(s)

Martin Hecht

---

checkData *Check Datasets for Missing Values and Invalid Codes*

Description

Check data frames for missing or duplicated entries in the ID variable, persons and/or variables without valid codes, and invalid codes. Invalid codes are codes which are not specified in table values.

Usage

```
checkData (dat, values, subunits, units)
```

Arguments

dat	A data frame
values	A data frame with code information. See 'Details'.
subunits	A data frame with subunit information. See 'Details'.
units	A data frame with unit information. See 'Details'.

Details

The results of checkData will be written to a protocol file with sunk.

Examples of data frames values, subunits and units can be found via data(inputList).

Value

Used for its side effects. The return value is NULL.

Author(s)

Nicole Haag, Anna Lenski

References

For missing types see http://code.google.com/p/zkdlib/wiki/MissingHandling

See Also

sunk

`checkInput` *Check Input Data Frames*

Description

Check input data frames for consistency and replace missing information with default values (if necessary).

Usage

```
checkInput(values, subunits, units, checkValues = TRUE, checkUnits = TRUE)
```

Arguments

`values`	A data frame with code information. See 'Details'
`subunits`	A data frame with subunit information. See 'Details'
`units`	A data frame with unit information. See 'Details'.
`checkValues`	Logical: Should data frame `values` be checked?
`checkUnits`	Logical: Should data frame `units` be checked?

Details

This function is largely for internal use and is called by `makeInputLists` before lists are generated.

Examples of data frames `values`, `subunits` and `units` can be found via `data(inputList)`.

Value

A list containing the checked and (if necessary) defaulted input data frames:

`values`	Checked data frame with code information. Will be returned if `checkValues = TRUE`.
`subunits`	A data frame with subunit information.
`units`	A data frame with unit information. Will be returned if `checkUnits = TRUE`.

Warning

Function will not check input data frames if `checkValues` and `checkUnits` are both `FALSE`.

Author(s)

Nicole Haag

See Also

makeInputLists

---

checkLink *checkLink*

Description

checks whether items in a dataset are linked via design

Usage

```
checkLink ( dataFrame, sysmis = NA, verbose = TRUE)
```

Arguments

dataFrame	A data.frame where all columns denote test items
sysmis	character string specifying values to be treat as missing by design
verbose	logical: Should output printed to console?

Value

A logical value, i.e. TRUE or FALSE, indicating whether items in dataset are linked to each other.

Author(s)

Sebastian Weirich

---

collapseMissings *Collapse Missings*

Description

converts character missings of different types to 0 or NA

Usage

```
collapseMissings(dat, missing.rule = NULL, item.names)
```

Arguments

dat	data frame containing character missings (e.g. type 'mbd' - missing by design)
missing.rule	list, definition how to recode distinct missings in dataset. See details for default.
item.names	character vector containing column names of the data frames whose character missings are to be collapsed

Details

Default `missing.rule` in `collapseMissings` is: text volume insufficient = 0 , missing not reached = 0 , missing coding impossible = NA , missing by design = NA , missing invalid response = 0 , missing by intention = 0

The results of `collapseMissings` will be written to a protocol file with `sunk`.

Value

A data frame with recoded missings.

Author(s)

Karoline Sachse, Martin Hecht

References

For missing types see http://code.google.com/p/zkdlib/wiki/MissingHandling

Examples

```
data(inputDat)
dat1 <- inputDat[[1]]  # get first dataset from inputDat
datColMis <- collapseMissings(dat = dat1,
missing.rule = list(mvi = 0 ,mnr = 0 ,mci = 0 ,mbd = NA ,mir = 0 ,mbi = 0),
item.names=colnames(dat1)[- c(1:2)])
```

---

commonItems *identify common items of groups*

Description

This function identifies items that groups of persons have in common.

Usage

```
commonItems ( data , group.var , missing = NA , uncommon = FALSE , simplify = TRUE )
```

Arguments

`data`	data.frame
`group.var`	group variable in data.frame , eihter numeric indicator of column or column name
`missing`	missing specification
`uncommon`	if `TRUE` a vector of uncommon items is additionally returned
`simplify`	if `TRUE` a character vector is returned (only in case of 2 groups and uncommon=`FALSE`)

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

names of list are both group names concatenated by "|"

Author(s)

Martin Hecht

Examples

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

# common items ar listed for each combination of groups
str ( commonItems ( data = d , group.var = "version" , missing = "mbd" ) )

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

`crop` *crop*

Description

remove trailing and leading characters from character strings

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

`detect.suppression` *detect suppression effects in regression models*

Description

This function detects suppression effects in regression models.

Usage

```
detect.suppression ( data , dependent , independent , full.return = FALSE , xlsx.path = NULL )
```

Arguments

`data`	data.frame with data to be used
`dependent`	dependent variable in regression model
`independent`	character vector of independent variables in regression model
`full.return`	if `FALSE` a data.frame as a quadratic matrix with suppression effects (TRUE/FALSE) of independent variables is returned
	if `TRUE` a data.frame with all calculated terms ist returned
`xlsx.path`	full path of Excel file that results should be written to

Details

formulae (13.39a) and (13.39b) decribed in Bortz (1999) page 446 are used

if `full.return=TRUE` a data.frame is returned.

Columns are:

rownames: <dependent variable> ~ <independent variables> | <independent variable that is tested for suppression>

multiple.reg: logical, indicates wether there are 2 (FALSE) or more than 2 (TRUE) independent variables in the regression model

dep: dependent variabel in regression model

pred: independent variable that is investigated on suppression effect

preds: independent variables in regression model besides `pred`

cor_pred_c: correlation of `pred` and dependent variable

cor_pred_fitted_c: correlation of predicted `pred` by indepenent variables and dependent variable

r.sq_pred: R squared from model predicting `pred` by independent variables

rterm.minus: right term in formula (13.39a)

rterm.plus: right termn in formula (13.39b)

rterm.minus.diff: difference of `rterm.minus` and `cor_pred_c`

rterm.plus.diff: difference of `cor_pred_c` and `rterm.plus`

(positive difference of `rterm.minus.diff` or `rterm.plus.diff` indicates suppression effect)

rterm.minus.log: logical value of formula (13.39a)

rterm.plus.log: logical value of formula (13.39b)

suppression: logical, `rterm.minus.log` | `rterm.plus.log`

if `full.return=FALSE` a data.frame as quadratic matrix is returned:

rows and columns are independent variables

diagonal includes `suppression` for suppression effect of variable in multiple regression

triangles include `suppression` for bivariate independent variables, "row" suppresses "column"

Value

depends on options `full.return`

Author(s)

Martin Hecht

References

for formulae used by `detect.suppression` see

Bortz, J. (1999). Statistik fuer Sozialwissenschaftler. 5. Auflage. Berlin: Springer. p. 446

`dichotomize` *dichotomize a numeric vector*

Description

dichotomize a numeric vector by median or mean split

Usage

```
dichotomize ( numvec , method = c("median","mean") , randomize = TRUE , ... )
```

Arguments

`numvec`	numeric vector
`method`	either median or mean split
`randomize`	logical, if `TRUE` elements that equal the split threshold are randomly assigned to one of the two groups if `FALSE` default behavior of cut is used
`...`	arguments are passed to set.seed and cut

Value

returns vector with dichotomization indicators

Author(s)

Martin Hecht

Examples

```
numvec <- c(1,2,3,4,5)
dichotomize ( numvec )

# set seed for random assignment of elements that match split threshold by passing argument 'seed' to funct
# ( '3' in numvec is on threshold if median is used )
dichotomize ( numvec , seed = 12345 )

# set level names by passing argument 'labels' to cut function
dichotomize ( numvec , labels = c ( "low" , "high") )
```

---

exploreDesign *explore data design*

Description

explore data structure with respect to specific missing code (e.g. "missing by design")

Usage

```
exploreDesign ( data , missing = NA , id = NULL , itemsPerPerson = TRUE , personsPerItem = TRUE
```

Arguments

data	data.frame
missing	missing specification
id	id variable in data if exists
itemsPerPerson	logical , if TRUE items per person list is returned
personsPerItem	logical , if TRUE persons per item list is returned

Value

depends on itemsPerPerson and personsPerItem , if both are TRUE a list with both elements is returned

Author(s)

Martin Hecht

Examples

```
data(science1)
d <- science1[,!colnames(science1) %in% science1.context.vars]
design <- exploreDesign ( data = d , missing = "mbd" , id = "id" )
str(design)
```

get.dsc *Reads Conquest descriptive files.*

Description

Reads Conquest files comprising descriptive population statistics generated by the 'descriptives' statement.

Usage

```
get.dsc(file)
```

Arguments

file	Character string of the Conquest descriptives file.

Value

A list of n elements, with n the number of groups in the analysis. Each element is a list with two data frames, the single and aggregated descriptives of the corresponding group. Single descriptives give for each dimension the number of observations, mean, standard deviation and variance of the corresponding estimate, i.e. the WLE or the plausible values (PVs). When descriptives for PVs are read in, mean, standard deviation and variance for each pausible value is given. Aggregated descriptives give mean, standard deviation and variance of the standard error of corresponding estimate. When descriptives vor plausible values are read in, aggregated descriptives gibt also mean, standard deviation and variance of all plausible values.

References

See Conquest manual, pp.162.

get.equ *Reads equivalence table created in Conquest analysis.*

Description

Reads Conquest files comprising equivalence tables for MLE or WLE parameters.

Usage

```
get.equ(file.equ)
```

Arguments

file.equ	Character string of the Conquest equ-file.

Value

A list of n+1 elements, with n the number of dimensions in the analysis. Each element is a data.frame, whose name correponds to the name of the dimension the values belongs to. All data.frames except the last one give the transformation of each possible raw score to the WLE or MLE score including it's standard error. First column in each data.frame contains the raw score, second column the transformed WLE or MLE score, third columns it's standard error.

The last element of the list give some sparse information about the model specifications.

References

See Conquest manual, pp.162.

`get.itn` *get.itn*

Description

blablabla

Usage

```
get.itn(file)
```

Arguments

`file`

`get.plausible` *Reads Conquest plausible values files*

Description

Function reads Conquest plausible value files and transforms them into a R data frame.

Usage

```
get.plausible(file)
```

Arguments

`file`	Character string of the Conquest plausible values file to be read in.

Details

Funktion identifies number of cases, number of plausible values and number of dimensions.

Value

A data frame, where each row corresponds to one case. Columns are labeled with dimension names and number of corresponding plausible value.

case	Case number. Each row represents one person.
ID	Case ID, if listed in Conquest plausible values file.
pv	Plausible value. Denotation of columns names is pv.[name of dimension]_[number of plausible value]. For example, pv.reading_6 refers to the 6th plausible value of reading dimension.
eap	Expectation value of the a posterio distribution of the corresponding dimension.
eap.se	Standard error of the EAP estimate.

---

get.q3 *get.q3*

Description

get Q3 statistics

Usage

```
get.q3 ( results )
```

Arguments

results	results (structured list) from automateModels run

Value

list (analyses) of data.frames in matrix format containing Q3 statistics

Author(s)

Martin Hecht

---

get.shw *Reads Conquest showfiles*

Description

Function reads Conquest showfiles and transforms them into a R list of data frames.

Usage

```
get.shw(file, dif.term = NULL, split.dif = TRUE,
        abs.dif.bound = 0.64, sig.dif.bound = 0.43)
```

Arguments

file	Character string of the Conquest showfile to be read in.
dif.term	Optional: Character string. Name of the term considered to be DIF-term. Must match corresponding term in showfile.
split.dif	Logical: When TRUE, DIF-Parameter are only given for Reference group.
abs.dif.bound	When DIF-Parameter are evaluated, this specifies the critical value for absolute DIF.
sig.dif.bound	When DIF-Parameter are evaluated, this specifies the critical value for confidence interval DIF.

Details

Funktion searches for 'TERM'-statements in Conquest showfile and reads the tables associated with. If one statement is specified to contain DIF analyses, absolute DIF value is computed 2*[group-specific parameter]. Confidence intervalls for 90, 95 and 99 percent are computed via the standard error of specific parameters. If both criteria - absolute DIF exceeds abs.dif.bound and the confidence intervall does not include sig.dif.bound, item is considered to have DIF.

Value

A list of data frames, named by the 'TERM'-statements in Conquest showfile, plus an additional data frame with regression coefficients when latent linear regression model was specified in Conquest analysis. If one term was specified as DIF-statement, the corresponding data frame is augmented with additional columns for confidence intervals and indicators specifying significant DIF.

Each data frame corresponding to a 'TERM' statement contains following columns:

item	Name of item
ESTIMATE	Estimated difficulty of item
ERROR	Standard error of estimated item difficulty
MNSQ	Item's 'Outfit'
MNSQ.1	Items's 'Infit'
CI	Lower and upper bound confidence intervals
T	T values, corresponding to confidence intervals
filename	Name of show file read in
abs.dif	Only for DIF analysis. Absolute DIF, computed as 2*[group-specific parameter].
ci.lb	Lower bound confidence interval for specific significance level of 90, 95 or 99 percent.
ci.ub	Upper bound confidence interval for specific significance level of 90, 95 or 99 percent.
sig	Indicates whether the corresponding item matches both DIF criteria. See details.

When latent regression was specified, the last element of the returned list is a data frame with regression coefficients, corresponding to the number of dimensions and the number of regressors. Regressor names, regression coefficients and its standard errors are given for each dimension.

Rows represent the regressors, columns represent the latent dimension to which the regression is fitted.

get.wle *Reads Conquest WLE or MLE files.*

Description

Reads Conquest files comprising maximum likelihood estimates (MLE) or weighted likelihood estimates (WLE).

Usage

```
get.wle(file)
```

Arguments

file	Character string of the Conquest MLE or WLE file to be read in.

Value

A data frame with columns according to the corresponding MLE or WLE file. For each dimension of the analysis number of solved items, number of presented items, point estimate and its standard error is given. Each row represents one person. Columns are named as follows:

case	Case number. Each row represents one person.
n.solved	Number of solved items by the i-th person.
n.total	Number of total items presented to the i-th person.
wle	WLE or MLE estimate.
wle.se	Standard error of WLE or MLE estimate.

The last number of columns names represents the dimension the WLE or MLE estimate belongs to.

getConquestVersion *get version (build) of ConQuest*

Description

get version (build) of ConQuest

Usage

```
getConquestVersion ( path.conquest , asDate = TRUE )
```

Arguments

path.conquest	full path to ConQuest executable console
asDate	if TRUE an object of class 'date' is returned if FALSE a character string is returned

Value

depends on option 'asDate'

Author(s)

Martin Hecht

Examples

```
getConquestVersion ( "c:/ConQuest/console_Feb2007.exe" )
```

---

inputDat *List of Three Datasets from Educational Assessment*

Description

Simulated data for three booklets for an educational assessment study.

Usage

```
data(inputDat)
```

Format

This list contains 3 data frames, each with the following columns:

ID Person-ID

Hisei A continuous covariate.

Ixx Item responses to a selection of 30 test items.

Details

code, subunit and unit descriptions are stored in dataset `inputList`.

Examples

```
data(inputDat)
str(inputDat)
```

---

inputList *Data Frames with Code, Subunit and Unit Information for Datasets in inputDat*

Description

These data frames contain information about codes, subunits and units for the datasets in `inputDat` and are necessary inputs for functions `automateDataPreparation`, `checkData`, `recodeData` and `aggregateData`.

Usage

```
data(inputList)
```

Format

A list with three data frames:

1. units: Unit information, contains the following columns:

   unit Unit name.

   unitType Subunit types: `ID` = ID variable; `TI` = test item; `CV` = context variable.

   unitLabel Unit label, to be used by `writeSpss`.

   unitDescription Unit description.

   unitAggregateRule Aggregate rule for unit: `SUM`; `MEAN`.

   unitScoreRule Scoring rule for unit (not sure how this will be used in the future.)

2. subunits: Subunit information, contains the following columns:

   unit Unit name, for which subunits are given.

   subunit Subunit name.

   subunitType Subunit types:`'?'`.

   subunitLabel Subunit label, to be used by `writeSpss`.

   subunitDescription Subunit descriptions.

   subunitPosition Subunit position in test booklet (e.g., line 1).

   subunitTransniveau Subunit transformation level.

   subunitRecoded Name of recoded subunit.

   subunitLabelRecoded Label for recoded subunit, to be used when `writeSpss` is applied to a dataset produced by `recodeData`.

3. values: Value information, contains the following columns:

   subunit Subunit name, for which values are given.

   value Valid values for the respective subunit.

   valueRecode Recode values for the respective value.

   valueType Value types: `vc` = valid code; `mbd` = missing – by design; `mvi` = missing – volume insufficient; `mnr` = missing – not reached; `mci` = missing – coding impossible; `mbi` = missing – by intention.

   valueLabel Value labels, to be used by `writeSpss`.

   valueDescription Value descriptions.

   valueLabelRecoded Labels for recoded values, to be used when `writeSpss` is applied to a dataset produced by `recodeData`.

   valueDescriptionRecoded Descriptions for recoded values.

4. unitRecodings: Unit recoding information, contains the following columns:

   unit Unit name

   value Valid values for the respective unit.

   valueRecode Recode values for the respective value.

   valueType Value types: `vc` = valid code; `mbd` = missing – by design; `mvi` = missing – volume insufficient; `mnr` = missing – not reached; `mci` = missing – coding impossible; `mbi` = missing – by intention.

   valueLabel Value labels, to be used by `writeSpss`.

   valueDescription Value descriptions.

   valueLabelRecoded Labels for recoded values, to be used when `writeSpss` is applied to a dataset produced by `recodeData`.

5. savFiles: information for `loadSav`, contains the following columns:

filename SPSS filenames

case.id ID variable in the respective dataset, used by `mergeData`

6. newID: information for `mergeData`, contains the following columns:

   key one of the entries should be `master-id`

   value the corresponding value; how the ID variable in the final dataset shall be named

7. aggrMiss: missing aggregation pattern for `aggregateData`

Examples

```
data(inputList)
str(inputList)
```

`loadSav` *loadSav*

Description

read SPSS data files and change id names, if necessary

Usage

```
loadSav(path = getwd(), savFiles = NULL, oldIDS, newID, correctDigits = FALSE, truncateSpaceChar
```

Arguments

`path`

`savFiles`

`oldIDS`

`newID`

`correctDigits`

`truncateSpaceChar`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ( path=getwd(), savFiles=NULL, oldIDS, newID, correctDigits=FALSE, truncateSpaceChar = TRUE ) {
           funVersion <- "loadSAV_0.0.2"
           if(missing(oldIDS))  {stop(paste("Error in ",funVersion,": 'oldIDS' is missing.\n",sep="")) }
           if(missing(newID))   {stop(paste("Error in ",funVersion,": 'newID' is missing.\n",sep="")) }
           if(length(newID)!=1) {stop(paste("Error in ",funVersion,": 'newID' has to be of length 1.\n",sep
           # if(!exists("read.spss"))  {library(foreign)}
           if(!is.null(savFiles)) {
              fileExists <- file.exists(file.path(path,savFiles))
              if(all(!fileExists)) {
                 stop(paste("Error in ",funVersion,": None of the files specified in 'savFiles' were found
```

```
   }
   if(!all(fileExists)) {
      cat(paste(funVersion,": Following files specified in 'savFiles' were not found in ",path,'
      notFoundFiles <- savFiles[!fileExists]
      FoundFiles    <- savFiles[fileExists]
      cat(paste(notFoundFiles,collapse=", "))
      cat("\nOnly found files will be read in.\n")
      savFiles      <- savFiles[fileExists]
   }
}
if(is.null(savFiles)) {
   savFiles <- list.files(path=path,pattern=".sav|.SAV",recursive=FALSE)
   if(length(savFiles)==0) {
      stop(paste("No '.sav'-files found in ",path,".\n",sep=""))
   }
cat(paste(funVersion,": Found ", length(savFiles), " 'savFiles' in ",path,".\n",sep=""))
}
### hier beginnt das eigentliche Einlesen
allDataFrames <- NULL
for (i in seq(along=savFiles)) {
     file.i <- data.frame(read.spss(file.path(path,savFiles[i]),to.data.frame=FALSE, use.value.l
     idCol  <- unique(unlist(lapply(oldIDS, FUN=function(ii) {grep(ii,colnames(file.i))})))
     if(length(idCol)<1) {
        stop(paste("Error in ",funVersion,": None of the specified 'oldIDS' were found in datase
     }
     if(length(idCol)>1) {
        stop(paste("Error in ",funVersion,": More than one of the specified 'oldIDS' were found
     }
     colnames(file.i)[idCol] <- newID
     ### Leerzeichen abschnipseln
     if(truncateSpaceChar == TRUE)  {
        for (ii in 1:ncol(file.i))  {
             file.i[,ii] <- crop(file.i[,ii])
        }
     }
     ### Stelligkeitskorrektur
     if(correctDigits == TRUE) {
        colsToCorrect <- lapply(1:ncol(file.i), FUN=function(ii) { sort(unique(nchar(file.i[,ii]
        options(warn = -1)
        colsToCorrect <- which( unlist( lapply(colsToCorrect,  FUN=function(ii) { all(ii == c(1,
        options(warn = 0)
        if(length(colsToCorrect)>0) {
           cat(paste(funVersion,": ",length(colsToCorrect)," columns are corrected for column wi
           for (ii in colsToCorrect) {
                file.i[,ii] <- gsub(" ","0", formatC(as.character(file.i[,ii]),width=2))
           }
        }
     }
     allDataFrames[[i]] <- file.i
}
return(allDataFrames)
}
```

---

long2matrix *long2matrix*

Description

transforms long format data.frame into a matrix format data.frame

Usage

```
long2matrix ( data , sort = TRUE , triangle = NULL ,
force.diagonal = FALSE , exclude.diagonal = FALSE ,
long2matrix = TRUE )
```

Arguments

data	data.frame with columns "row" , "col" , "val"
sort	sort rows and columns of matrix
triangle	if not NULL a symmetric matrix will be constructed available options are "upper" , "lower" , "both"
force.diagonal	a diagonal is forced into matrix even if no diagonal elements are in data
exclude.diagonal	the diagonal is excluded if possible
long2matrix	if FALSE data is not transformed

Value

long2matrix = TRUE
: data.frame in matrix format

long2matrix = FALSE
: data.frame in long format

Author(s)

Martin Hecht

Examples

```
d1 <- data.frame (
"row" = c ( "v1" , "v2" , "v2" , "v3" , "v1" , "v3" ) ,
"col" = c ( "v1" , "v3" , "v2" , "v1" , "v2" , "v3" ) ,
"val" = c ( 1 , 5 , 4 , 3 , 2 , 6 ) , stringsAsFactors = FALSE )

# unsorted matrix
long2matrix  ( data = d1 , sort = FALSE )
# sorted by default
long2matrix  ( data = d1 )
# extract upper triangle of symmetric matrix
long2matrix  ( data = d1 , triangle = "upper" )
# exclude diagonal elements
long2matrix  ( data = d1 , triangle = "upper" , exclude.diagonal = TRUE )
# if full matrix ("both" triangles) is requested, the diagonal cannot be excluded, option is ignored
long2matrix  ( data = d1 , triangle = "both" , exclude.diagonal = TRUE )

# no diagonal elements are specified
d2 <- data.frame (
"row" = c ( "v2" , "v1" , "v1" ) ,
"col" = c ( "v3" , "v3" , "v2" ) ,
"val" = c ( 5 , 3 , 2 ) , stringsAsFactors = FALSE )
```

```
long2matrix ( data = d2 )
# diagonal is set (with NAs)
long2matrix ( data = d2 , triangle = "upper" , force.diagonal = TRUE )
```

---

makeCodebookInput *Make Input Data Frames From IQB-Codebooks*

Description

Make Input Data Frames From IQB-Codebooks

Usage

```
makeCodebookInput(codebook)
```

Arguments

codebook	dataframe IQB-Codebook

Details

xxx

Value

xxx

---

makeInputLists *Generate Input Lists for Functions* checkData, recodeData *and* aggregateData

---

Description

Transforms information given in values, subunits and units in a format that is used by checkData, recodeData and aggregateData.

Usage

```
makeInputLists(values, subunits, units, recodedData = TRUE)
makeInputCheckData(values, subunits, units)
makeInputRecodeData(values, subunits)
makeInputAggregateData(subunits, units, recodedData = TRUE)
```

Arguments

values	A data frame with code information. See Details.
subunits	A data frame with subunit information. See Details.
units	A data frame with unit information. See Details.
recodedData	Logical indicating whether colnames in dataset to aggregate are the subunit names (as in subunits$subunit) or recoded subunit names (as in subunits$subunitRecoded). Default is TRUE, meaning that colnames are recoded subitem names. This parameter is only relevant when input for aggregateData is generated.

Details

This function generates specific inputs for the data preparation functions checkData, recodeData and aggregateData. It is largely for internal use of these functions, who call their respective version.

Examples of data frames values, subunits and units can be found via data(inputLists).

Value

A list with several of the following entries (depending on which version of the function is called):

varinfoRaw	A list with information about variables and their values expected in raw data.
varinfoRecoded	A list with information about variables and their values expected in recoded data.
varinfoAggregated	A list with information about variables and their values expected in aggregated data.
recodeinfo	A list with information needed for recoding of data.
aggregateinfo	A list with information needed for aggregation of data.

Author(s)

Nicole Haag

Examples

```
data(inputList)
lists <- makeInputLists(inputList$values, inputList$subunits, inputList$units, recodedData = TRUE)
str(lists)
```

---

makeNumeric *Change Character Variables to* numeric

Description

Converts character variables, which contain only values, to numeric. Character variables containing letters are not converted. This avoids warnings, if conversion to numeric is attempted for variables, which contain characters.

Usage

```
makeNumeric(variable)
```

Arguments

variable	Variable to be changed to numeric.

Value

Variable converted to numeric, if possible.

Author(s)

Nicole Haag

Examples

```
a <- c("1", "2", "3", "4")
b <-  c("1", "2", "x", "4")
makeNumeric(a)
makeNumeric(b)
```

---

mergeData *Merge Data Frames using one Key Variable*

Description

Merges several data frames and matches them using one key variable

Usage

```
mergeData(newID = "ID", datList, oldIDs=NULL, addMbd = FALSE, writeLog=FALSE)
```

Arguments

newID	character string containing the key variable's name in the merged dataset
datList	list of data frames to be merged
oldIDs	character vector OR numeric vector containing either names of the key variables in datList or their column number in each dataframe in datList default is a vector containing replicates of the value of newID.
addMbd	logical; string "mbd" (missing by desgin) will be added instead of NA
writeLog	logical; if Logfile shall be written via sunk.

Value

A data frame containing unique cases and unique variables. All cases and all variables that could be identified the original data frames will be kept and matched.

Author(s)

Karoline Sachse, Nicole Haag

Examples

```
data(inputDat)
str(inputDat)

mergedDataset <- mergeData("person-id", inputDat, c("idstud", "idstud", "idstud"), addMbd=TRUE)
str(mergedDataset)

mergedDataset <- mergeData("idstud", inputDat, writeLog=FALSE)
str(mergedDataset)
```

prepare.package *prepare.package*

Description

prepares package

Usage

```
prepare.package ( source.folder = "p:/ZKD/development" ,
files ,
package.folder = "p:/ZKD/packages" ,
package.name ,
package.version )
```

Arguments

source.folder	folder of R files
files	character vector of R files that should be included in the package
package.folder	folder of packages
package.name	name of package
package.version	version of package, must be in format "0.0.0"

Details

copies R files from source.folder to package folder copies "x.x.x" folder content to package folder modifies version and date in DESCRIPTION and automateModels creates ChangeLog

Author(s)

Martin Hecht

readDaemonXlsx *read xlsx-Files produced by ZKDaemon*

Description

read xlsx-Files produced by ZKDaemon

Usage

```
readDaemonXlsx(filename)
```

Arguments

filename	A character string containing path, name and extension of .xlsx produced by ZKDaemon. Caution! Sheet order is important (see Datails).

Details

Compulsory: 1st sheet: units. 2nd sheet: subunits. 3rd sheet: values. Optional: 4th sheet: unitRecodings. 5th sheet: savFiles. 6th sheet: newID. 7th sheet: aggregateMissings. 8th sheet: unitProperties. 9th sheet: property labels. 10th sheet: booklets.

Value

A list of data frames containing information that is required by `automateDataPreparation`

Author(s)

Karoline Sachse

Examples

```
str(inputList)
```

---

recodeData *Recode Datasets with Missing Values*

Description

Recode datasets with special consideration of missing values.

Usage

```
recodeData(dat, values, subunits)
```

Arguments

`dat`	A data frame
`values`	A data frame with code information. See 'Details'.
`subunits`	A data frame with subunit information. See 'Details'.

Details

`recodeData` recodes data frames with special consideration of missing values. The results of `recodeData` will be written to a protocol file with `sunk`. `recodeData` will give warnings, if missing or incomplete recode informations are found. Values without recode information will NOT be recoded!

Examples of data frames `values` and `subunits` can be found via `data(inputList)`

Value

A data frame with recoded variables according to the specifications in `values` and `subunits`. Colnames will be the names specified in `subunits$subunitRecoded`.

Author(s)

Martin Hecht, Christiane Penk, Nicole Haag

References

http://code.google.com/p/zkdlib/wiki/MissingHandling

See Also

aggregateData, checkData

Examples

```
data(inputDat)
data(inputList)
# library(car)

dat1 <- inputDat[[1]]  # get first dataset from inputDat
datRec <- recodeData(dat1, inputList$values, inputList$subunits)
str(datRec)
```

---

reinsort.col *reinsort.col*

Description

insert columns of dataframe in specific position

Usage

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

Arguments

dat	data.frame on which operation should be performed
toreinsort	column name(s) or numeric indicator(s) that should be relocated
after	column name or numeric indicator after that toreinsort should be located

Value

data.frame

Author(s)

Martin Hecht

rmNA *remove NA columns and rows from data*

Description

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

Usage

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

Arguments

data	data.frame or matrix
remove	if TRUE columns and rows are removed, if FALSE a list of identified columns and rows is returned
verbose	if TRUE removed columns and rows are printed on output window

Value

depends on option remove

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,N
( rmNA ( mat , verbose = TRUE ) )
( rmNA ( mat , remove = FALSE ) )
```

rmNAcols *remove NA columns from data*

Description

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

Usage

```
rmNAcols ( data , rows = NULL , tolerance = 0 , cumulate = TRUE , remove = TRUE , verbose = FALS
```

Arguments

data	data.frame or matrix
rows	rows to include, 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 removed columns and rows are printed on output window

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,N

# 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( c(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 ( data , cols = NULL , tolerance = 0 , cumulate = TRUE , remove = TRUE , verbose = FALS
```

Arguments

data	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 ) , nc

# 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.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

set.col.type *set type of variable in data.frame*

Description

converts type of column(s) to "character" , "numeric" , "logical" , "integer" or "factor"

Usage

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

Arguments

data	data.frame
col.type	named list of variable names that are to be converted. names of list is conversion type ("character" , "numeric" , "numeric.if.possible" , "logical" , "integer" or "factor")
verbose	if TRUE variables that have been converted are printed
...	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

Author(s)

Martin Hecht

Examples

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

---

sortDfrByNames *sort data.frame by colnames and/or rownames*

Description

specify new colnames and/or rownames order, data.frame is sorted in accordance

Usage

```
sortDfrByNames ( dfr , col.order = NULL , row.order = NULL , warn = TRUE )
```

Arguments

dfr	data.frame
col.order	character vector of colnames in new order
row.order	character vector of rownames in new order
warn	logical, if TRUE warnings are printed on output window if col.order/row.order do not correspond to colnames/rownames resp.

Value

data.frame

Author(s)

Martin Hecht

Examples

```
dfr <- data.frame ( matrix ( rnorm ( 100 ) , ncol = 10 ) )
colnames ( dfr ) <- paste ( "X" , 10:1 , sep = "" )
rownames ( dfr ) <- paste ( "X" , 11:2 , sep = "" )
dfr

# sort data.frame by 'col.order' and 'row.order'
sortDfrByNames ( dfr , paste ( "X" , 1:10 , sep = "" ) , paste ( "X" , 2:11 , sep = "" ) )
```

source.it.all *source.it.all*

Description

sources *.R files of folder

Usage

```
source.it.all ( folder="p:/ZKD/development" , develop.modules = NULL , return.stable = FALSE )
```

Arguments

folder	folder with *.R files
develop.modules	character vector of R files that should be sourced in development status
return.stable	if TRUE nothing is sourced and a vector of all stable versions is returned

Value

return.stable = FALSE	sources R files
return.stable = TRUE	character vector of stable R files

Author(s)

Christiane Penk, Martin Hecht

source.it.all2 *source.it.all2*

Description

sources *.R files of folder

Usage

```
source.it.all2 ( folder="p:/ZKD/development" , use.zkd.conv = TRUE , development = TRUE , develo
```

Arguments

folder	folder with *.R files
development	if TRUE development versions are sourced (if non-existent the latest stable is sourced or nothing is sourced, see option development.only\ if FALSE stable versions are sourced
use.zkd.conv	if TRUE R files in folder are checked to be consisten with specific ("zkd") versioning convention \ if FALSE all R files in folder are sourced
development.only	if TRUE only development versions are sourced \ if FALSE stable versions are included
exclude	character vector of R files that should not be sourced

Value

sources R files

Author(s)

Martin Hecht, Christiane Penk

---

sunk *sunk*

Description

writes output to file

Usage

```
sunk ( cmd = NULL , path = NULL , write = TRUE , console.output = TRUE , new.file = FALSE , text
```

Arguments

cmd	character string of element to write, may be either text (e.g. "write me to file") or a function call (e.g. "summary(lm)")
path	path (folder and name) to output file if NULL path is defaulted to getwd()+"sunk.txt" all environments are searched for sunk.path, if sunk.path is found (exists), it is used
write	logical, if TRUE (default) output is written to file
console.output	logical, if TRUE (default) output is displayed on console
new.file	logical, if TRUE the output file is created if FALSE (default) output is appended to existing file
text.on.error	logical, sunk checks if the character string 'cmd' is an evaluatable expression if TRUE (default), 'cmd' is treated as text if an error occurs when trying to evaluate string if FALSE, sunk stops on errors/not evaluatable expressions
text.out.method	choose "cat" (default) or "print" as the output method for text

Value

writes to disk

Author(s)

Martin Hecht

writeSpss *Export Datasets to SPSS*

Description

Writes data and SPSS syntax files.

Usage

```
writeSpss(dat, values, subunits, units,
filedat = "zkddata.txt", filesps = "readZkdData.sps",
missing.rule = list(mvi = 0, mnr = 0, mci = NA, mbd = NA, mir = 0, mbi = 0),
        path = getwd(), sep = "\t", dec = ",", silent = FALSE)
```

Arguments

dat	A data frame
values	A data frame with code information. See 'Details'.
subunits	A data frame with subunit information. See 'Details'.
units	A data frame with unit information. See 'Details'.
filedat	A character string with the name of the output data file.
filesps	A character string with the name of the output syntax file.
missing.rule	A list containing recode information for character missings. See 'References' for description of default values.
path	A character string containing the path of the output file. The value in path is appended to filedat and filesps. By default, files are written to the current R working directory. If path=NULL then no file path appending is done.
sep	The separator between the data fields.
dec	The decimal separator for numerical data.
silent	A logical flag stating whether the names of the files should be printed.

Details

This function automates most of the work needed to export a dataset to SPSS. It uses a modified version of writeForeignSPSS() from the foreign() package and of mids2spss() fromt the mice package. The modified version allows for a choice of the field and decimal separators, makes some improvements to the formatting and provides variable labels and value labels according to the information in the data frames values, subunits and units.

Examples of data frames values, subunits and units can be found on data(inputList)

The SPSS syntax file has the proper file names and separators set, so in principle it should run and read the data without alteration. SPSS is more strict than R with respect to the paths. Always use the full path, otherwise SPSS may not be able to find the data file.

Value

Used for its side effects. The return value is NULL.

Author(s)

Nicole Haag

References

http://code.google.com/p/zkdlib/wiki/MissingHandling

`yen.q3` *yen.q3*

Description

Q3 statistics

Usage

```
yen.q3 ( dat , theta , b , progress = T )
```

Arguments

`dat`	bla
`theta`	bla
`b`	bla
`progress`	bla

Index