Package 'eatGADS'

February 23, 2021

Title Data Management of Large Hierarchical Data

Version 0.16.0

Description Import 'SPSS' data, handle and change 'SPSS' meta data, store and access large hierarchical data in 'SQLite' data bases.

Depends R (>= 3.5.0)

Imports eatDB (>= 0.4.1), haven (>= 2.1.0), plyr, eatTools (>= 0.4.0), tibble, data.table, hms

License GPL (>= 2)

URL https://github.com/beckerbenj/eatGADS

Encoding UTF-8

LazyData true

Suggests testthat, knitr, rmarkdown, covr, tidyr (>= 1.1.0)

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Benjamin Becker [aut, cre]

Maintainer Benjamin Becker <b.becker@iqb.hu-berlin.de>

Repository CRAN

Date/Publication 2021-02-23 19:30:02 UTC

R topics documented:

`applyChangeMeta` *Apply Meta Data Changes.*

Description

Function to apply meta data changes to a `GADSdat` object specified by a change table extracted by `getChangeMeta`.

Usage

```
applyChangeMeta(changeTable, GADSdat, ...)

## S3 method for class 'varChanges'
applyChangeMeta(changeTable, GADSdat, ...)

## S3 method for class 'valChanges'
applyChangeMeta(
  changeTable,
  GADSdat,
  existingMeta = c("stop", "value", "value_new"),
  ...
)
```

Arguments

`changeTable`	Change table as provided by `getChangeMeta`.
`GADSdat`	`GADSdat` object imported via `eatGADS`.
`...`	further arguments passed to or from other methods.
`existingMeta`	If values are recoded, which meta data should be used (see details)?

Details

Values for which the change columns contain `NA` remain unchanged. If changes are performed on value levels, recoding into existing values can occur. In these cases, `existingMeta` determines how the resulting meta data conflicts are handled, either raising an error if any occur (`"stop"`), keeping the original meta data for the value (`"value"`) or using the meta data in the `changeTable` or, if incomplete, from the recoded value (`"value_new"`).

Value

Returns the modified `GADSdat` object.

Examples

```
# Change a variable name and label
varChangeTable <- getChangeMeta(pisa, level = "variable")
varChangeTable[1, c("varName_new", "varLabel_new")] <- c("IDstud", "Person ID")

pisa2 <- applyChangeMeta(varChangeTable, GADSdat = pisa)
```

---

applyLookup *Recode via lookup table.*

Description

Recode one or multiple variables based on a lookup table created via createLookup (and potentially formatted by collapseColumns).

Usage

```
applyLookup(GADSdat, lookup, suffix = NULL)
```

Arguments

GADSdat	A GADSdat object.
lookup	Lookup table created by createLookup and - if necessary - collapsed by collapseColumns. Column names must be c("variable","value","value_new").
suffix	Suffix to add to the existing variable names. If NULL, the old variables will be overwritten.

Details

If there are missing values in the column value_new, NAs are inserted as new values and a warning is issued.

The complete work flow when using a lookup table to recode multiple variables in a GADSdat could be: (0) optional: Recode empty strings to NA (necessary, if the look up table is written to excel). (1) create a lookup table with createLookup. (2) Save the lookup table to .xlsx with write_xlsx from eatAnalysis. (3) fill out the lookup table via Excel. (4) Import the lookup table back to R via read_excel from readxl. (5) Apply the final lookup table with applyLookup.

See applyLookup_expandVar for recoding a single variable into multiple variables.

Value

Returns a recoded GADSdat.

Examples

```
## create an example GADSdat
iris2 <- iris
iris2$Species <- as.character(iris2$Species)
gads <- import_DF(iris2)

## create Lookup
lu <- createLookup(gads, recodeVars = "Species")
lu$value_new <- c("plant 1", "plant 2", "plant 3")

## apply lookup table
gads2 <- applyLookup(gads, lookup = lu, suffix = "_r")

## only recode some values
lu2 <- createLookup(gads, recodeVars = "Species")
lu2$value_new <- c("plant 1", "plant 2", NA)
gads3 <- applyLookup(gads, lookup = lu2, suffix = "_r")
```

---

applyLookup_expandVar *Recode via lookup table into multiple variables.*

Description

Recode one or multiple variables based on a lookup table created via createLookup. In contrast to applyLookup, this function allows the creation of multiple resulting variables from a single input variable. All variables in lookup except variable and value are treated as recode columns.

Usage

```
applyLookup_expandVar(GADSdat, lookup)
```

Arguments

GADSdat	A GADSdat object.
lookup	Lookup table created by createLookup.

Details

If a variable contains information that should be split into multiple variables via manual recoding, applyLookup_expandVar can be used. If there are missing values in any recode column, NAs are inserted as new values. A warning is issued only for the first column.

The complete work flow when using a lookup table to expand variables in a GADSdat based on manual recoding could be: (1) create a lookup table with createLookup. (2) Save the lookup table to .xlsx with write_xlsx from eatAnalysis. (3) fill out the lookup table via Excel. (4) Import the lookup table back to R via read_excel from readxl. (5) Apply the final lookup table with applyLookup_expandVar.

See applyLookup for simply recoding variables in a GADSdat.

Value

Returns a recoded GADSdat.

Examples

```
## create an example GADSdat
example_df <- data.frame(ID = 1:6,
                         citizenship = c("germ", "engl", "germ, usa", "china",
                                         "austral, morocco", "nothin"),
                         stringsAsFactors = FALSE)
gads <- import_DF(example_df)

## create Lookup
lu <- createLookup(gads, recodeVars = "citizenship", addCol = c("cit_1", "cit_2"))
lu$cit_1 <- c("German", "English", "German", "Chinese", "Australian", NA)
lu$cit_2 <- c(NA, NA, "USA", NA, "Morocco", NA)

## apply lookup table
gads2 <- applyLookup_expandVar(gads, lookup = lu)
```

---

applyNumCheck *Apply recodes according to a numerical check data.frame.*

Description

Applies recodes as specified by a numCheck data.frame, as created by createNumCheck.

Usage

```
applyNumCheck(GADSdat, numCheck)
```

Arguments

GADSdat	A GADSdat object.
numCheck	A data.frame as created by createNumCheck.

Details

This function is currently under development.

Value

A recoded GADSdat.

Examples

```
# tbd
```

changeMissings *Change missing code.*

Description

Change or add missing codes of a variable as part of a GADSdat or all_GADSdat object.

Usage

```
changeMissings(GADSdat, varName, value, missings)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of a variable name.
value	Numeric values.
missings	Character string of the new missing codes, either "miss" or "valid".

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta.

Value

Returns the GADSdat object with changed meta data.

Examples

```
# Set a specific value to missing
pisa2 <- changeMissings(pisa, varName = "computer_age",
                        value = 5, missings = "miss")

# Set multiple values to missing
pisa3 <- changeMissings(pisa, varName = "computer_age",
                        value = 1:4,
                        missings = c("miss", "miss", "miss", "miss"))

# Set a specific value to not missing
pisa4 <- changeMissings(pisa2, varName = "computer_age",
                        value = 5, missings = "valid")
```

changeSPSSformat *Change SPSS format.*

Description

Change the SPSS format of a variable as part of a GADSdat or all_GADSdat object.

Usage

```
changeSPSSformat(GADSdat, varName, format)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of variable names.
format	A single string containing the new SPSS format, for example 'A25' or 'F10'.

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta.

Value

Returns the GADSdat object with changed meta data..

Examples

```
pisa2 <- changeSPSSformat(pisa, varName = "idstud",
                          format = "F10.0")
```

changeValLabels *Change value labels.*

Description

Change or add value labels of a variable as part of a GADSdat or all_GADSdat object.

Usage

```
changeValLabels(GADSdat, varName, value, valLabel)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of a variable name.
value	Numeric values.
valLabel	Character string of the new value labels.

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta.

Value

Returns the GADSdat object with changed meta data.

Examples

```
# Change existing value labels
pisa2 <- changeValLabels(pisa, varName = "repeated",
                         value = c(1, 2),
                         valLabel = c("no grade repetition", "grade repitition"))

# Add value label to unlabeled value
mtcars_g <- import_DF(mtcars)
mtcars_g2 <- changeValLabels(mtcars_g, varName = "cyl",
                             value = c(4, 6, 8),
                             valLabel = c("four", "six", "eight"))
```

---

changeVarLabels *Change the variable label.*

Description

Change the variable label of a variable as part of a GADSdat or all_GADSdat object.

Usage

```
changeVarLabels(GADSdat, varName, varLabel)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of variable names.
varLabel	Character string of the new variable labels.

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta.

Value

Returns the GADSdat object with changed meta data.

Examples

```
# Change one variable label
pisa2 <- changeVarLabels(pisa, varName = "repeated",
                         varLabel = c("Has a grade been repeated?"))
```

---

changeVarNames *Change Variable Names.*

Description

Change variable names of a GADSdat or all_GADSdat object.

Usage

```
changeVarNames(GADSdat, oldNames, newNames)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
oldNames	Vector containing the old variable names.
newNames	Vector containing the new variable names, in identical order as oldNames.

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta

Value

Returns the GADSdat object with changed variable names.

Examples

```
# Change multiple variable name
pisa2 <- changeVarNames(pisa, oldNames = c("idstud", "idschool"),
                         newNames = c("IDstud", "IDschool"))
```

checkEmptyValLabels *Check Value Labels*

Description

Check value labels for (a) value labels with no occurrence in the data `checkEmptyValLabels` and (b) values with no value labels `checkMissingValLabels`.

Usage

```
checkEmptyValLabels(GADSdat, vars = namesGADS(GADSdat), valueRange = NULL)

checkMissingValLabels(GADSdat, vars = namesGADS(GADSdat), valueRange = NULL)
```

Arguments

`GADSdat`	A `GADSdat` object.
`vars`	Character vector with the variable names to which `checkValLabels()` should be applied.
`valueRange`	[optional] Numeric vector of length 2: In which range should numeric values be checked? If specified, only numeric values are returned and strings are omitted.

Details

`NA`s are excluded from this check. Designated missing codes are reported normally.

Value

Returns a list of length 2. Each contains a list of the length of `vars`.

`labels with no values`
: Value labels with no occurrence in the data

`not labeled values`
: Values in the data with no value labels

Functions

- `checkEmptyValLabels`: check for superfluous value labels
- `checkMissingValLabels`: check for missing value labels

Examples

```
# Check a categorical and a metric variable
checkMissingValLabels(pisa, vars = c("g8g9", "age"))
checkEmptyValLabels(pisa, vars = c("g8g9", "age"))

# Check while defining a specific value range
checkMissingValLabels(pisa, vars = c("g8g9", "age", "idschool"),
```

```
              valueRange = c(0, 5))
checkEmptyValLabels(pisa, vars = c("g8g9", "age", "idschool"),
              valueRange = c(0, 5))
```

---

checkLEStructure *Checks compatibility of GADS data bases with a linking error data base.*

Description

This function checks if a linking error data base is compatible with the two trend eatGADS data bases. For checking the compatibility of two eatGADS data bases see checkTrendStructure.

Usage

```
checkLEStructure(filePath1, filePath2, lePath)
```

Arguments

filePath1	Path of the first eatGADS .db file.
filePath2	Path of the second eatGADS .db file.
lePath	Path of the linking error eatGADS .db file.

Details

This function inspects whether all linking error variables correspond to variables in the eatGADS data base and if the key variables also correspond to existing variables in the trend eatGADS data bases.

Value

Returns a report list.

---

checkMissings *Check and Adjust Missing Coding*

Description

Function to check if missings are coded and labeled correctly in a GADSdat object.

Usage

```
checkMissings(
  GADSdat,
  missingLabel = "missing",
  addMissingCode = TRUE,
  addMissingLabel = FALSE
)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
missingLabel	Single string indicating how missing labels are commonly named in the value labels.
addMissingCode	If TRUE, missing codes are added according to occurrence of "missingLabel" in "valLabel".
addMissingLabel	If TRUE, "generic missing" is added according to occurrence of "miss" in "missings". As often various value labels for missings are used, this argument should be used with great care.

Details

The function compares value labels "valLabels" and missing codes "missings" of a GADSdat object and its meta data information. Mismatches are reported and can be automatically adjusted.

Value

Returns a GADSdat object.

Examples

```
# Change example data set (create a value label with incorrect missing code)
pisa2 <- changeValLabels(pisa, varName = "computer_age",
                         value = 5, valLabel = "missing: No computer use")

pisa3 <- checkMissings(pisa2)
```

---

checkTrendStructure *Checks compatibility of two* eatGADS *data bases.*

Description

This function checks if both data bases perform identical joins via foreign keys, if they contain the same variable names and if these variables have the same value labels. Results of this comparison are reported on data table level as messages and as an output list.

Usage

```
checkTrendStructure(filePath1, filePath2)
```

Arguments

filePath1	Path of the first eatGADS .db file.
filePath2	Path of the second eatGADS .db file.

Details

An error is thrown if the key structure or the data table structure differs between the two data bases. Differences regarding meta data for missing value labels and for variables labels (and formatting) are ignored.

Value

Returns a report list.

---

checkValue *Check for a specific value*

Description

Function to look for occurrences of a specific value in a GADSdat.

Usage

```
checkValue(GADSdat, value, vars = namesGADS(GADSdat))
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
value	Single string indicating how missing labels are commonly named in the value labels.
vars	Character vector with the variable names to which checkValue should be applied.

Details

The function checks occurrences of a specific value in a set of variables (default: all variables) in the GADSdat and outputs a vector containing the count of occurrences for all variables in which the value occurs. It explicitly supports checking for NA.

Value

A named integer.

Examples

```
# for all variables in the data
checkValue(pisa, value = 99)

# only for specific variables in the data
checkValue(pisa, vars = c("idschool", "g8g9"), value = 99)
```

---

checkVarNames *Check names for* SQLite *conventions.*

Description

Applies variable names changes to GADSdat or all_GADSdat objects.

Usage

```
checkVarNames(GADSdat)
```

Arguments

GADSdat	GADSdat or all_GADSdat object imported via eatGADS.

Details

Illegal names in a SQLite data base include SQLite keywords (see sqlite_keywords) and names with a "." in it.

Value

Returns the original object with updated variable names.

Examples

```
# Change example data set (create an invalid variable name)
pisa2 <- changeVarNames(pisa, oldNames = "computer_age",
                        newNames = "computer.age")

pisa3 <- checkVarNames(pisa2)
```

clean_cache *Clean temporary cache.*

Description

Deprecated. The cached data base is now cleaned when the R sessions ends automatically.

Usage

```
clean_cache(tempPath = tempdir())
```

Arguments

tempPath	Local directory in which the data base was temporarily be stored.

Details

Cleans the temporary cache, specified by `tempdir()`. This function had to be executed at the end of an R session if `getGADS_fast` or `getTrendGADS` with `fast = TRUE` had been used.

Value

Returns nothing.

collapseColumns *Collapse two columns of a lookup table.*

Description

Collapse two columns or format a single column of a lookup table created by `createLookup`.

Usage

```
collapseColumns(lookup, recodeVars, prioritize)
```

Arguments

lookup	For example a lookup table `data.frame` as created via `createLookup`.
recodeVars	Character vector of column names which should be collapsed (currently only up to two variables are supported).
prioritize	Character vector of length 1. Which of the columns in `recodeVars` should be prioritized, if multiple values are available? If `recodeVars` is of length 1, this argument can be omitted.

Details

If a lookup table is created by createLookup, different recoding columns can be specified by the addCols argument. This might be the case if two rater suggest recodes or one rater corrects recodes by another rater in a separate column. After the recoding columns have been filled out, collapseColumns can be used to either:

(a) Collapse two recoding columns into one recoding column. This might be desirable, if the two columns contain missing values. prioritize can be used to specify, which of the two columns should be prioritized if both columns contain valid values.

(b) Format the lookup table for applyLookup, if recodeVars is a single variable. This simply renames the single variable specified under recodeVars.

Value

Returns a data.frame that can be used for applyLookup, with the columns:

variable	Variable names
value	Old values
value_new	New values. Renamed and/or collapsed column.

Examples

```
## (a) Collapse two columns
# create example recode data.frame
lookup_raw <- data.frame(variable = c("var1"), value = c("germa", "German", "dscherman"),
           recode1 = c(NA, "English", "German"),
           recode2 = c("German", "German", NA), stringsAsFactors = FALSE)

# collapse columns
lookup <- collapseColumns(lookup_raw, recodeVars = c("recode1", "recode2"), prioritize = "recode2")

## (b) Format one column
# create example recode data.frame
lookup_raw2 <- data.frame(variable = c("var1"), value = c("germa", "German", "dscherman"),
           recode1 = c("German", "German", "German"), stringsAsFactors = FALSE)

# collapse columns
lookup2 <- collapseColumns(lookup_raw2, recodeVars = c("recode1"))
```

---

collapseMC_Text *Recode a multiple choice variable according to a character variable.*

Description

Recode an labeled integer variable (based on an multiple choice item), according to a character variable (e.g. an open answer item).

Usage

```
collapseMC_Text(
  GADSdat,
  mc_var,
  text_var,
  mc_code4text,
  var_suffix = "_r",
  label_suffix = "(recoded)"
)
```

Arguments

GADSdat	A GADSdat object.
mc_var	The variable name of the multiple choice variable.
text_var	The variable name of the text variable.
mc_code4text	The value label in mc_var that indicates that information from the text variable should be used.
var_suffix	Variable name suffix for the newly created variables. If NULL, variables are overwritten.
label_suffix	Variable label suffix for the newly created variable (only added in the meta data). If NULL no suffix is added.

Details

Multiple choice variables can be represented as labeled integer variables in a GADSdat. Multiple choice items with a forced choice frequently contain an open answer category. However, sometimes open answers overlap with the existing categories in the multiple choice item. collapseMC_Text allows recoding the multiple choice variable based on the open answer variable.

mc_code4text indicates when entries in the text_var should be used. Additionally, entries in the text_var are also used when there are missings on the mc_var. New values for the mc_var are added in the meta data, while preserving the initial ordering of the value labels. Newly added value labels are sorted alphabetically.

For more details see the help vignette: vignette("recoding_forcedChoice",package = "eatGADS").

Value

Returns a GADSdat containing the newly computed variable.

Examples

```
# Example gads
example_df <- data.frame(ID = 1:5, mc = c("blue", "blue", "green", "other", "other"),
                         open = c(NA, NA, NA, "yellow", "blue"),
                         stringsAsFactors = FALSE)
example_df$mc <- as.factor(example_df$mc)
gads <- import_DF(example_df)
```

```
# recode
gads2 <- collapseMC_Text(gads, mc_var = "mc", text_var = "open",
                         mc_code4text = "other")
```

`collapseMultiMC_Text` *Recode multiple choice variable with multiple variables.*

Description

Recode multiple variables (representing a single multiple choice item) based on multiple character variables (representing a text field).

Usage

```
collapseMultiMC_Text(
  GADSdat,
  mc_vars,
  text_vars,
  mc_var_4text,
  var_suffix = "_r",
  label_suffix = "(recoded)",
  invalid_miss_code = -98,
  invalid_miss_label = "Missing: Invalid response",
  notext_miss_code = -99,
  notext_miss_label = "Missing: By intention"
)
```

Arguments

`GADSdat`	A `GADSdat` object.
`mc_vars`	A character vector with the variable names of the multiple choice variable. Names of the character vector are the corresponding values that are represented by the individual variables. Creation by `matchValues_varLabels` is recommended.
`text_vars`	A character vector with the names of the text variables which should be collapsed.
`mc_var_4text`	The name of the multiple choice variable that signals that information from the text variable should be used. This variable is recoded according to the final status of the text variables.
`var_suffix`	Variable suffix for the newly created `GADSdat`. If an empty character, the existing variables are overwritten.
`label_suffix`	Suffix added to variable label for the newly created or modified variables in the `GADSdat`.
`invalid_miss_code`	Missing code which is given to new character variables if all text entries where recoded into the dichotomous variables.

`invalid_miss_label`
: Value label for `invalid_miss_code`.

`notext_miss_code`
: Missing code which is given to empty character variables.

`notext_miss_label`
: Value label for `notext_miss_code`.

Details

If a multiple choice item can be answered with ticking multiple boxes, multiple variables in the data set are necessary to represent this item. In this case, an additional text field for further answers can also contain multiple values at once. However, some of the answers in the text field might be redundant to the dummy variables. `collapseMultiMC_Text` allows to recode multiple MC items of this kind based on multiple text variables. The recoding can be prepared by expanding the single text variable (`createLookup` and `applyLookup_expandVar`) and by matching the dummy variables to its underlying values stored in variable labels (`matchValues_varLabels`).

The function recodes the dummy variables according to the character variables. Additionally, the `mc_var_4text` variable is recoded according to the final status of the `text_vars` (exception: if the text variables were originally `NA`, `mc_var_4text` is left as it was).

Missing values in the character variables can be represented either by `NA`s or by empty characters. The multiple choice variables specified with `mc_vars` can only contain the values `0`, `1` and missing codes. The value `1` must always represent "this category applies". If necessary, use `recodeGADS` for recoding.

For cases for which the `text_vars` contain only values that can be recoded into the `mc_vars`, all new `text_vars` are given specific missing codes (see `invalid_miss_code` and `invalid_miss_label`). All remaining `NA`s on the character variables are given a specific missing code (`notext_miss_code`).

Value

Returns a `GADSdat` containing the newly computed variables.

Examples

```
# Prepare example data
mt2 <- data.frame(ID = 1:4, mc1 = c(1, 0, 0, 0), mc2 = c(0, 0, 0, 0), mc3 = c(0, 1, 1, 0),
                  text1 = c(NA, "Eng", "Aus", "Aus2"), text2 = c(NA, "Franz", NA, "Ger"),
                   stringsAsFactors = FALSE)
mt2_gads <- import_DF(mt2)
mt3_gads <- changeVarLabels(mt2_gads, varName = c("mc1", "mc2", "mc3"),
                            varLabel = c("Lang: Eng", "Aus spoken", "other"))

## All operations (see also respective help pages of functions for further explanations)
mc_vars <- matchValues_varLabels(mt3_gads, mc_vars = c("mc1", "mc2", "mc3"),
            values = c("Aus", "Eng", "Eng"), label_by_hand = c("other" = "mc3"))

out_gads <- collapseMultiMC_Text(mt3_gads, mc_vars = mc_vars,
             text_vars = c("text1", "text2"), mc_var_4text = "mc3")

out_gads2 <- multiChar2fac(out_gads, vars = c("text1_r", "text2_r"))
```

```
final_gads <- remove2NAchar(out_gads2, vars = c("text1_r_r", "text2_r_r"),
                      max_num = 1, na_value = -99, na_label = "missing: excessive answers")
```

---

compareGADS *Compare two GADS.*

Description

Compare multiple variables of two GADSdat or all_GADSdat objects.

Usage

```
compareGADS(
  GADSdat_old,
  GADSdat_new,
  varNames,
  output = c("list", "data.frame", "aggregated")
)
```

Arguments

GADSdat_old	GADSdat object imported via eatGADS.
GADSdat_new	GADSdat object imported via eatGADS.
varNames	Character string of variable names to be compared.
output	How should the output be structured?

Details

Returns "all equal" if the variable is identical across the objects or a data.frame containing a frequency table with the values which have been changed. Especially useful for checks after recoding.

Value

Returns either a list with "all equal" and data.frames or a single data.frame.

Examples

```
# Recode a GADS
pisa2 <- recodeGADS(pisa, varName = "schtype",
                        oldValues = 3, newValues = 9)
pisa2 <- recodeGADS(pisa2, varName = "language",
                        oldValues = 1, newValues = 15)

# Compare
compareGADS(pisa, pisa2,
            varNames = c("ganztag", "schtype", "language"), output = "list")
```

```
compareGADS(pisa, pisa2,
            varNames = c("ganztag", "schtype", "language"), output = "data.frame")
compareGADS(pisa, pisa2,
             varNames = c("ganztag", "schtype", "language"), output = "aggregated")
```

---

createGADS *Create an* eatGADS *data base.*

Description

Creates a relational data base containing hierarchically stored data with meta information (e.g. value and variable labels).

Usage

```
createGADS(allList, pkList, fkList, filePath)
```

Arguments

allList	An object created via mergeLabels.
pkList	List of primary keys.
fkList	List of foreign keys.
filePath	Path to the db file to write (including name); has to end on '.db'.

Details

Uses createDB from the eatDB package to create a relational data base. For details on how to define keys see the documentation of createDB.

Value

Creates a data base in the given path, returns NULL.

Examples

```
# see createDB vignette
```

createLookup *Extract values for recoding.*

Description

Extract unique values from one or multiple variables of a GADSdat object for recoding (e.g. via an Excel spreadsheet).

Usage

```
createLookup(GADSdat, recodeVars, sort_by = NULL, addCols = c("value_new"))
```

Arguments

GADSdat	A GADSdat object.
recodeVars	Character vector of variable names which should be recoded.
sort_by	By which column (variable and/or value) should the long format data.frame be sorted? If NULL, no sorting is performed.
addCols	Character vector of additional column names for recoding purposes.

Details

If recoding of one or multiple variables is more complex, a lookup table can be created for later application via applyLookup or applyLookup_expandVar. The function allows the extraction of the values of multiple variables and sorting of these unique values via variable and/or values. If addCols are specified the lookup table has to be formatted via collapseColumns, before it can be applied to recode data.

Value

Returns a data frame in long format with the following variables:

variable	Variables as specified in recodeVars
value	Unique values of the variables specified in recodeVars
value_new	This is the default for addCols. If different additional column names are supplied, this column is missing.

Examples

```
# create example GADS
dat <- data.frame(ID = 1:4, var1 = c(NA, "Eng", "Aus", "Aus2"),
                  var2 = c(NA, "French", "Ger", "Ita"),
                  stringsAsFactors = FALSE)
gads <- import_DF(dat)

# create Lookup table for recoding
lookup <- createLookup(gads, recodeVars = c("var1", "var2"), sort_by = c("value", "variable"))
```

```
# create Lookup table for recoding by multiple recoders
lookup2 <- createLookup(gads, recodeVars = c("var1", "var2"), sort_by = c("value", "variable"),
                        addCols = c("value_recoder1", "value_recoder2"))
```

---

createNumCheck *Create data.frame for specification of numerical plausibility checks.*

Description

All numerical variables without value labels in a GADSdat are selected and a data.frame is created, which allows the specification of minima and maxima.

Usage

```
createNumCheck(GADSdat)
```

Arguments

GADSdat	A GADSdat object.

Details

This function is currently under development.

Value

A data.frame with the following variables:

variable	All numerical variables in the GADSdat
varLabel	Corresponding variable labels
min	Minimum value for the specific variable.
max	Maximum value for the specific variable.
value_new	Which value should be inserted if values exceed the specified range?

Examples

```
# tbd
```

eatGADS *eatGADS: Data management of hierarchical SPSS files via R and SQLite*

Description

The eatGADS package provides various groups of functions: importing data (mainly sav-files), handling and modifying meta data on variable level, creating a fixed form SQLite data base and using the SQLite data base.

Importing data

SPSS data can be imported via import_spss, R data.frames via import_DF.

Creating the GADS

Hierarchical data sets are combined via mergeLabels and the data base is created via createGADS. For this, the package eatDB is utilized. See also createDB.

Using the GADS

The content of a data base can be obtained via namesGADS. Data is extracted from the data base via getGADS for a single GADS and via getTrendGADS for trend analysis. The resulting object is a GADSdat object. Meta data can be extracted via extractMeta, either from the GADSdat object or directly from the data base. Data can be extracted from the GADSdat object via extractData.

export_tibble *Transform a* GADSdat *to a* tibble

Description

haven's read_spss stores data together with meta data (e.g. value and variable labels) in a tibble with attributes on variable level. This function transforms a GADSdat object to such a tibble.

Usage

```
export_tibble(GADSdat)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.

Details

This function is mainly intended for internal use.

Value

Returns a tibble.

Examples

```
pisa_tbl <- export_tibble(pisa)
```

---

extractData *Extract Data*

Description

Extract data.frame from a GADSdat object for analyses in R. For extracting meta data see extractMeta.

Usage

```
extractData(
  GADSdat,
  convertMiss = TRUE,
  convertLabels = "character",
  dropPartialLabels = TRUE,
  convertVariables
)
```

Arguments

GADSdat	A GADSdat object.
convertMiss	Should values coded as missing values be recoded to NA?
convertLabels	If "numeric", values remain as numerics. If "factor" or "character", values are recoded to their labels. Corresponding variable type is applied.
dropPartialLabels	Should value labels for partially labeled variables be dropped? If TRUE, the partial labels will be dropped. If FALSE, the variable will be converted to the class specified in convertLabels.
convertVariables	Character vector of variables names, which labels should be applied to. If not specified (default), value labels are applied to all variables for which labels are available. Variable names not in the actual GADS are silently dropped.

Details

A GADSdat object includes actual data (GADSdat$dat) and the corresponding meta data information (GADSdat$labels). extractData extracts the data and applies relevant meta data (missing conversion, value labels), so the data can be used for analyses in R.

If factor are extracted via convertLabels == "factor", the underlying integers will be tried to preserved. If this is not possible, a warning is issued. As SPSS has almost no limitations regarding the underlying values of labeled integers and R's factor format is very strict (no 0, only integers increasing by + 1), this procedure can lead to frequent problems.

Value

Returns a data frame.

Examples

```
# Extract Data for Analysis
dat <- extractData(pisa)

# convert labeled variables to factors
dat <- extractData(pisa, convertLabels = "factor")

# convert only some variables to factor
dat <- extractData(pisa, convertLabels = "factor", convertVariables = c("schtype", "ganztag"))
```

---

extractGADSdat *Extract single* GADSdat *from* all_GADSdat

Description

Function to extract a single GADSdat from an all_GADSdat object.

Usage

```
extractGADSdat(all_GADSdat, name)
```

Arguments

all_GADSdat	all_GADSdat object
name	A character vector with length 1 with the name of the GADSdat

Details

GADSdat objects can be merged into a single all_GADSdat object via mergeLabels. This function, performs the reverse action, extracting a single GADSdat object.

Value

Returns an GADSdat object.

Examples

```
# see createGADS vignette
```

extractMeta *Get Meta Data*

Description

Extract meta data (e.g. variable and values labels) from an eatGADS object. This can be a GADSdat, an all_GADSdat, a labels data.frame, or the path to an existing data base.

Usage

```
extractMeta(GADSobject, vars = NULL)
```

Arguments

GADSobject	Either a GADSdat object or a path to an existing eatGADS data base.
vars	A character vector containing variable names. If NULL (default), all available meta information is returned.

Details

Meta data is stored tidily in all GADSdat objects as a separate long format data frame. This information can be extracted for a single or multiple variables.

Value

Returns a long format data frame with meta information.

Examples

```
# Extract Meta data from data base
db_path <- system.file("extdata", "pisa.db", package = "eatGADS")
extractMeta(db_path, vars = c("schtype", "sameteach"))

# Extract Meta data from loaded/imported GADS
extractMeta(pisa, vars = c("schtype", "sameteach"))
```

extractVars *Extract or remove variables from a* GADSdat.

Description

Extract or remove variables and their meta data from a GADSdat object.

Usage

```
extractVars(GADSdat, vars)

removeVars(GADSdat, vars)
```

Arguments

GADSdat	GADSdat object.
vars	A character vector containing the variables names in the GADSdat.

Details

Both functions simply perform the variable removal or extraction from the underlying data.frame in the GADSdat object followed by calling updateMeta.

Value

Returns a GADSdat object.

Examples

```
## create an example GADSdat
example_df <- data.frame(ID = 1:4,
                         age = c(12, 14, 16, 13),
                         citizenship1 = c("German", "English", "Polish", "Chinese"),
                         citizenship2 = c(NA, "German", "Chinese", "Polish"),
                         stringsAsFactors = TRUE)
gads <- import_DF(example_df)

## remove variables from GADSdat
gads2 <- removeVars(gads, vars = c("citizenship2", "age"))

## extract GADSdat with specific variables
gads3 <- extractVars(gads, vars = c("ID", "citizenship1"))
```

---

getChangeMeta *Extract table for Meta Data Changes.*

Description

Function to obtain a data frame from a GADSdat object for for changes to meta data on variable or on value level.

Usage

```
getChangeMeta(GADSdat, level = "variable")
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
level	'variable' or 'value'.

Details

Changes on variable level include variable names (varName), variable labels (varLabel), SPSS format ((format)) and display width (display_width). Changes on value level include values (value), value labels (valLabel) and missing codes (missings).

Value

Returns the meta data sheet for all variables including the corresponding change columns.

Examples

```
# For changes on variable level
varChangeTable <- getChangeMeta(pisa, level = "variable")

# For changes on value level
valChangeTable <- getChangeMeta(pisa, level = "value")
```

---

getGADS *Get data from GADS data base.*

Description

Extracts variables from a GADS data base. Only the specified variables are extracted. Note that this selection determines the format of the data.frame that is extracted.

Usage

```
getGADS(vSelect = NULL, filePath)
```

Arguments

vSelect	Character vector of variable names.
filePath	Path of the existing eatGADS data base file.

Details

See createDB and dbPull for further explanation of the query and merging processes.

Value

Returns a GADSdat object.

Examples

```
# Use data base within package
db_path <- system.file("extdata", "pisa.db", package = "eatGADS")
pisa_gads <- getGADS(db_path, vSelect = c("schtype", "sameteach"))
```

`getGADS_fast` *Get data from GADS data base fast from server directory.*

Description

Extracts variables from a `eatGADS` data base. Only the specified variables are extracted. Note that this selection determines the format of the `data.frame` that is extracted. CAREFUL: This function uses a local temporary directory to speed up loading the data base from a server and caches the data base locally for a running R session. The temporary data base is removed automatically when the running `R` session is terminated.

Usage

```
getGADS_fast(vSelect = NULL, filePath, tempPath = tempdir())
```

Arguments

`vSelect`	Character vector of variable names.
`filePath`	Path of the existing `eatGADS` data base file.
`tempPath`	Local directory in which the data base can temporarily be stored. Using the default is recommended.

Details

A random temporary directory is used for caching the data base and is removed, when the `R` sessions terminates. See `createDB` and `dbPull` for further explanation of the query and merging processes.

Value

Returns a `GADSdat` object.

getTrendGADS *Get data for trend reports.*

Description

Extracts variables from two eatGADS data bases and a linking error data base. Data can then be extracted from the GADSdat object via extractData. For extracting meta data from a data base or a GADSdat object see extractMeta. To speed up the data loading, getGADS_fast is used per default.

Usage

```
getTrendGADS(
  filePath1,
  filePath2,
  lePath = NULL,
  vSelect = NULL,
  years,
  fast = TRUE,
  tempPath = tempdir()
)
```

Arguments

filePath1	Path of the first eatGADS db file.
filePath2	Path of the second eatGADS db file.
lePath	Path of the linking error db file. If NULL, no linking errors are added to the data.
vSelect	Variables from both GADS to be selected (as character vector).
years	A numeric vector of length 2. The first elements corresponds to filePath1, the second element to filePath2.
fast	Should getGADS_fast be used for data loading instead of getGADS? Using the default is heavily recommended.
tempPath	The directory, in which both GADS will be temporarily stored. Using the default is heavily recommended.

Details

This function extracts data from two GADS data bases and a linking error data base. All data bases have to be created via createGADS. The two GADS are joined via rbind and a variable year is added, corresponding to the argument years. If lePath is specified, linking errors are also extracted and then merged to the GADS data. Make sure to also extract the key variables necessary for merging the linking errors (the domain variable for all linking errors, additionally the competence level variable for linking errors for competence levels). The GADSdat object can then further be used via extractData. See createDB and dbPull for further explanation of the querying and merging processes.

Value

Returns a GADSdat object.

Examples

```
# See getGADS vignette
```

`import_convertLabel` *Import an object imported via* `convertLabel`

Description

Function to import a data.frame object created by convertLabel for use in eatGADS. If possible, importing data via import_spss should always be preferred.

Usage

```
import_convertLabel(df, checkVarNames = TRUE)
```

Arguments

df	A data.frame.
checkVarNames	Should variable names be checked for violations of SQLite and R naming rules?

Details

convertLabel from R package eatAnalysis converts an object imported via read.spss (from the foreign package) to a data.frame with factors and variable labels stored in variable attributes.

Value

Returns a list with the actual data dat and a data frame with all meta information in long format labels.

`import_DF` *Import R* `data.frame`

Description

Function to import a `data.frame` object for use in `eatGADS` while extracting value labels from factors.

Usage

```
import_DF(df, checkVarNames = TRUE)
```

Arguments

`df`	A `data.frame`.
`checkVarNames`	Should variable names be checked for violations of `SQLite` and `R` naming rules?

Details

Factors are integers with labeled variable levels. `import_DF` extracts these labels and stores them in a separate meta data data.frame. See `import_spss` for detailed information.

Value

Returns a list with the actual data `dat` and a data frame with all meta information in long format `labels`.

Examples

```
dat <- import_DF(iris, checkVarNames = FALSE)

# Inspect Meta data
extractMeta(dat)

# Extract Data
dat <- extractData(dat, convertLabels = "character")
```

import_raw *Import R data frame with explicit meta data sheets*

Description

Function to import a data.frame object for use in eatGADS while adding explicit variable and value meta information through separate data.frames.

Usage

```
import_raw(df, varLabels, valLabels = NULL, checkVarNames = TRUE)
```

Arguments

df	A data.frame.
varLabels	A data.frame containing the variable labels. All variables in the data have to have exactly one column in this data.frame.
valLabels	A data.frame containing the value labels. All referenced variables have to appear in the data, but not all variables in the data have to receive value labels. Can be omitted.
checkVarNames	Should variable names be checked for violations of SQLite and R naming rules?

Details

The argument varLables has to contain exactly two variables, namely varName and varLabel. valLables has to contain exactly four variables, namely varName, value, valLabel and missings. The column value can only contain numerical values. The column missings can only contain the values "valid" and "miss". Variables of type factor are not supported in any of the data.frames.

Value

Returns a list with the actual data dat and with all meta information in long format labels.

Examples

```
dat <- data.frame(ID = 1:5, grade = c(1, 1, 2, 3, 1))
varLabels <- data.frame(varName = c("ID", "grade"),
                        varLabel = c("Person Identifier", "School grade Math"),
                        stringsAsFactors = FALSE)
valLabels <- data.frame(varName = c("grade", "grade", "grade"),
                        value = c(1, 2, 3),
                        valLabel = c("very good", "good", "sufficient"),
                        missings = c("valid", "valid", "valid"),
                        stringsAsFactors = FALSE)

gads <- import_raw(df = dat, varLabels = varLabels, valLabels = valLabels, checkVarNames = FALSE)

# Inspect Meta data
```

```
extractMeta(gads)

# Extract Data
dat <- extractData(gads, convertLabels = "character")
```

`import_raw2` *Import R data frame with a explicit meta data sheet*

Description

Function to create a `GADSdat` object based on a `dat` `data.frame` and a `labels` `data.frame`.

Usage

```
import_raw2(dat, labels)
```

Arguments

`dat`	A `dat` `data.frame` containing all actual data.
`labels`	A `labels` `data.frame` containing all meta data.

Details

A `GADSdat` is basically a `list` with two elements: a `dat` and a `labels` `data.frame`. If these elements are separated, they can be cleanly tied together again by `import_raw2`. The function performs extensive checks on the integrity of the resulting `GADSdat` object. See `import_spss` and `import_raw` for further details.

Value

Returns a `GADSdat` object.

Examples

```
dat <- data.frame(ID = 1:5, grade = c(1, 1, 2, 3, 1))
varLabels <- data.frame(varName = c("ID", "grade"),
                        varLabel = c("Person Identifier", "School grade Math"),
                        stringsAsFactors = FALSE)
valLabels <- data.frame(varName = c("grade", "grade", "grade"),
                        value = c(1, 2, 3),
                        valLabel = c("very good", "good", "sufficient"),
                        missings = c("valid", "valid", "valid"),
                        stringsAsFactors = FALSE)

gads <- import_raw(df = dat, varLabels = varLabels, valLabels = valLabels, checkVarNames = FALSE)

# separate the GADSdat object
dat <- gads$dat
```

```
labels <- gads$labels

# rejoin it
dat <- import_raw2(dat, labels)
```

`import_RDS` *Import* RDS *file*

Description

Function to import a `data.frame` stored as a `.RDS` file while extracting value labels from factors.

Usage

```
import_RDS(filePath, checkVarNames = TRUE)
```

Arguments

`filePath`	Source file location, ending on `.RDS`.
`checkVarNames`	Should variable names be checked for violations of `SQLite` and `R` naming rules?

Details

Factors are integers with labeled variable levels. `import_RDS` extracts these labels and stores them in a separate meta data data.frame. See `import_DF` for detailed information. This function is a wrapper around `import_DF`.

Value

Returns a list with the actual data `dat` and a data frame with all meta information in long format `labels`.

`import_spss` *Import SPSS data*

Description

Function to import `.sav` files while extracting meta information, e.g. variable and value labels.

Usage

```
import_spss(filePath, checkVarNames = TRUE, labeledStrings = FALSE)
```

Arguments

filePath	Source file location, ending on .sav.
checkVarNames	Should variable names be checked for violations of SQLite and R naming rules?
labeledStrings	Should strings as labeled values be allowed? This possibly corrupts all labeled values.

Details

SPSS files (.sav) store variable and value labels and assign specific formatting to variables. import_spss imports data from SPSS, while storing this meta-information separately in a long format data frame. Value labels and missing labels are used to identify missing values (see checkMissings). Time and date variables are converted to character.

Value

Returns a list with the actual data dat and a data frame with all meta information in long format labels.

Examples

```
# Use spss data from within package
spss_path <- system.file("extdata", "pisa.zsav", package = "eatGADS")
pisa_gads <- import_spss(spss_path)
```

---

import_stata *Import Stata data*

Description

Function to import .dta files while extracting meta information, e.g. variable and value labels.

Usage

```
import_stata(filePath, checkVarNames = TRUE, labeledStrings = FALSE)
```

Arguments

filePath	Source file location, ending on .dta.
checkVarNames	Should variable names be checked for violations of SQLite and R naming rules?
labeledStrings	Should strings as labeled values be allowed? This possibly corrupts all labeled values.

Details

Stata files (.dta) store variable and value labels and assign specific formatting to variables. import_stata imports data from Stata, while storing this meta-information separately in a long format data frame. Time and date variables are converted to character.

Value

Returns a list with the actual data `dat` and a data frame with all meta information in long format `labels`.

`labelsGADS` *Labels from relational* `eatGADS` *data base.*

Description

Returns the variable and value labels of all variables in the `eatGADS` data base.

Usage

```
labelsGADS(filePath)
```

Arguments

`filePath`	Path of the existing `eatGADS` data base.

Details

Variable, value and missing labels as stored in the original SPSS-files and factors from R files are converted to long format for storage in the data base. `labelsGADS` returns them as a long format data frame.

Value

Returns a long format data frame including variable names, labels, values, value labels and missing labels.

Examples

```
# Extract Meta data from data base
db_path <- system.file("extdata", "pisa.db", package = "eatGADS")
metaData <- labelsGADS(db_path)
```

matchValues_varLabels *Match regular expressions and variable names.*

Description

Using variable labels, matchValues_varLabels matches a vector of regular expressions to a set of variable names.

Usage

```
matchValues_varLabels(GADSdat, mc_vars, values, label_by_hand = character(0))
```

Arguments

GADSdat	A GADSdat object.
mc_vars	A vector containing the names of the variables, which should be matched according to their variable labels.
values	A character vector containing the regular expressions for which the varLabel column should be searched.
label_by_hand	Additional value - mc_var pairs. Necessary, if for some mc_vars no value exists.

Details

Multiple choice items can be stored as multiple dichotomous variables with the information about the variable stored in the variable labels. The function collapseMultiMC_Text can be used to collapse such dichotomous variables and a character variable, but requires a character vector with variables names of the multiple choice variables. matchValues_varLabels creates such a vector based on matching regular expressions (values) to variable labels.

Note that all variables in mc_vars have to be assigned exactly one value (and vice versa). If a variable name is missing in the output, an error will be thrown. In this case, the label_by_hand argument should be used to specify the regular expression variable name pair manually.

Value

Returns a named character vector. Values of the vector are the variable names in the GADSdat, names of the vector are the regular expressions.

Examples

```
# Prepare example data
mt2 <- data.frame(ID = 1:4, mc1 = c(1, 0, 0, 0), mc2 = c(0, 0, 0, 0), mc3 = c(0, 1, 1, 0),
                  text1 = c(NA, "Eng", "Aus", "Aus2"), text2 = c(NA, "Franz", NA, NA),
                  stringsAsFactors = FALSE)

mt2_gads <- import_DF(mt2)

mt3_gads <- changeVarLabels(mt2_gads, varName = c("mc1", "mc2", "mc3"),
```

```
                              varLabel = c("Lang: Eng", "Aus spoken", "other"))

out <- matchValues_varLabels(mt3_gads, mc_vars = c("mc1", "mc2", "mc3"),
                              values = c("Aus", "Eng", "Eng"),
                              label_by_hand = c("other" = "mc3"))
```

`merge.GADSdat` *Merge two* `GADSdat` *objects into a single* `GADSdat` *object.*

Description

Is a secure way to merge the data and the meta data of two `GADSdat` objects. Currently, only limited merging options are supported.

Usage

```
## S3 method for class 'GADSdat'
merge(x, y, by, all = TRUE, all.x = all, all.y = all, ...)
```

Arguments

`x`	`GADSdat` object imported via `eatGADS`.
`y`	`GADSdat` object imported via `eatGADS`.
`by`	A character vector.
`all`	A character vector (either a full join or an inner join).
`all.x`	See merge.
`all.y`	See merge.
`...`	Further arguments are currently not supported but have to be included for `R CMD` checks.

Details

If there are duplicate variables (except the variables specified in the `by` argument), these variables are removed from y. The meta data is joined for the remaining variables via `rbind`.

Value

Returns a `GADSdat` object.

mergeLabels *Prepare data and metadata*

Description

Transform multiple GADSdat objects into a list ready for data base creation.

Usage

```
mergeLabels(...)
```

Arguments

...	GADSdat objects, as named arguments in the correct merge order.

Details

The function createGADS takes multiple GADSdat objects as input. The function preserves the ordering in which the objects are supplied, which is then used for the merging order in createGADS. Additionally, the separate lists of meta information for each GADSdat are merged and a data frame unique identifier is added.

Value

Returns an all_GADSdat object, which consists of list with a list of all data frames "datList" and a single data frame containing all meta data information "allLabels".

Examples

```
# see createGADS vignette
```

miss2NA *Recode Missings to* NA

Description

Recode Missings to NA according to missing labels in label data.frame.

Usage

```
miss2NA(GADSdat)
```

Arguments

GADSdat	A GADSdat object.

Details

Missings are imported as their values via import_spss. Using the value labels in the labels data.frame, miss2NA recodes these missings codes to NA. This function is mainly intended for internal use.

Value

Returns a data.frame with NA instead of missing codes.

---

multiChar2fac *Transform multiple character variables to factors with identical levels.*

Description

Convert multiple character variables to factors, while creating a common set of value labels, which is identical across variables.

Usage

```
multiChar2fac(GADSdat, vars, var_suffix = "_r", label_suffix = "(recoded)")
```

Arguments

GADSdat	A data.frame or GADSdat object.
vars	A character vector with all variables that should be transformed to factor.
var_suffix	Variable suffix for the newly created GADSdat. If an empty character, the existing variables are overwritten.
label_suffix	Suffix added to variable label for the newly created variable in the GADSdat.

Details

If a set of variables has the same possible values, it is desirable that these variables share the same value labels, even if some of the values do not occur on the individual variables. This function allows the transformation of multiple character variables to factors while assimilating the value labels. The SPSS format of the newly created variables is set to F10.0.

If necessary, missing codes can be set after transformation via checkMissings for setting missing codes depending on value labels for all variables or changeMissings for setting missing codes for specific values in a specific variable.

Value

Returns a GADSdat containing the newly computed variable.

Examples

```
## create an example GADSdat
example_df <- data.frame(ID = 1:4,
                  citizenship1 = c("German", "English", "missing by design", "Chinese"),
                   citizenship2 = c("missing", "German", "missing by design", "Polish"),
                       stringsAsFactors = FALSE)
gads <- import_DF(example_df)

## transform multiple strings
gads2 <- multiChar2fac(gads, vars = c("citizenship1", "citizenship2"))

## set values to missings
gads3 <- checkMissings(gads2, missingLabel = "missing")
```

---

namesGADS *Variables names of a GADS.*

Description

Variables names of a GADSdat object, a all_GADSdat object or a eatGADS data base.

Usage

```
namesGADS(GADS)
```

Arguments

GADS	A GADSdat object, a all_GADSdat or the path to an existing eatGADS data base.

Details

If the function is applied to a GADSdat object, a character vector with all variable names is returned. If the function is applied to a all_GADSdat object or to the path of a eatGADS data base, a named list is returned. Each list entry represents a data table in the object.

Value

Returns a character vector or a named list of character vectors.

Examples

```
# Extract variable names from data base
db_path <- system.file("extdata", "pisa.db", package = "eatGADS")
namesGADS(db_path)

# Extract variable names  from loaded/imported GADS
namesGADS(pisa)
```

orderLike *Order the variables in a* GADSdat.

Description

Order the variables in a GADSdat according to a character vector. If there are discrepancies between the two sets, a warning is issued.

Usage

```
orderLike(GADSdat, newOrder)
```

Arguments

GADSdat	A GADSdat object.
newOrder	A character vector containing the order of variables.

Details

The variables in the dat and in the labels section are ordered. Variables not contained in the character vector are moved to the end of the data.

Value

Returns a GADSdat object.

pisa *PISA Plus Example Data*

Description

A small example data set from the German PISA Plus campus files as distributed by the Forschungsdatenzentrum,IQB.

Usage

```
pisa
```

Format

A data.frame with 500 rows and 133 variables, including:

idstud Person ID variable

idschool School ID variable

schtype School type

...

Source

Research Data Center at the Institute for Educational Quality Improvement (2020). Programme for International Student Assessment - Plus 2012, 2013 (PISA Plus 2012-2013) - Campus File (Version 1) [Data set]. Berlin: Institute for Educational Quality Improvement. doi: 10.5159/IQB_PISA_Plus_201213_CF_v1

`recode2NA` *Recode a value to* `NA`.

Description

Recode a value in multiple variables in a `GADSdat` to `NA`.

Usage

```
recode2NA(GADSdat, recodeVars = namesGADS(GADSdat), value = "")
```

Arguments

`GADSdat`	A `GADSdat` object.
`recodeVars`	Character vector of variable names which should be recoded.
`value`	Which value should be recoded to `NA`?

Details

If there are value labels given to the specified value, these are removed. Number of recodes per variable are reported.

If a data set is imported from `.sav` character variables frequently contain empty strings. Especially if parts of the data are written to `.xlsx` this can cause problems (e.g. as look up tables from `createLookup`), as most function which write to `.xlsx` convert empty strings to `NA`s. `recodeString2NA` can be used to recode all empty strings to `NA` beforehand.

Value

Returns the recoded `GADSdat`.

Examples

```
# create example GADS
dat <- data.frame(ID = 1:4, var1 = c("", "Eng", "Aus", "Aus2"),
                  var2 = c("", "French", "Ger", "Ita"),
                  stringsAsFactors = FALSE)
gads <- import_DF(dat)

# recode empty strings
gads2 <- recode2NA(gads)
```

```
# recode numeric value
gads3 <- recode2NA(gads, recodeVars = "ID", value = 1)
```

---

recodeGADS *Recode a labeled variable.*

Description

Recode a labeled variable as part of a GADSdat or all_GADSdat object.

Usage

```
recodeGADS(
  GADSdat,
  varName,
  oldValues,
  newValues,
  existingMeta = c("stop", "value", "value_new")
)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Name of the variable to be recoded.
oldValues	Vector containing the old values.
newValues	Vector containing the new values (in the respective order as oldValues).
existingMeta	If values are recoded, which meta data should be used (see details)?

Details

Applied to a GADSdat or all_GADSdat object, this function is a wrapper of getChangeMeta and applyChangeMeta. oldValues and newValues are matched by ordering in the function call.

If changes are performed on value levels, recoding into existing values can occur. In these cases, existingMeta determines how the resulting meta data conflicts are handled, either raising an error if any occur ("stop"), keeping the original meta data for the value ("value") or using the meta data in the changeTable or, if incomplete, from the recoded value ("value_new").

Missing values (NA) are supported in oldValues but not in newValues. For recoding values to NA see recode2NA instead. For recoding character variables, using lookup tables via createLookup is recommended. For changing value labels see changeValLabels.

Value

Returns a GADSdat.

Examples

```
# Example gads
example_df <- data.frame(ID = 1:5, color = c("blue", "blue", "green", "other", "other"),
                         animal = c("dog", "Dog", "cat", "hors", "horse"),
                         age = c(NA, 16, 15, 23, 50),
                         stringsAsFactors = FALSE)
example_df$animal <- as.factor(example_df$animal)
gads <- import_DF(example_df)

# simple recode
gads2 <- recodeGADS(gads, varName = "animal",
                    oldValues = c(3, 4), newValues = c(7, 8))
```

---

recodeString2NA *Recode a string to* NA.

Description

Deprecated, use recode2NA instead..

Usage

```
recodeString2NA(GADSdat, recodeVars = namesGADS(GADSdat), string = "")
```

Arguments

GADSdat	A GADSdat object.
recodeVars	Character vector of variable names which should be recoded.
string	Which string should be recoded to NA?

Value

Returns the recoded GADSdat.

---

remove2NAchar *Shorten multiple text variables while giving NA codes.*

Description

Shorten text variables from a certain number on while coding overflowing answers as complete missings.

Usage

```
remove2NAchar(GADSdat, vars, max_num = 2, na_value, na_label)
```

Arguments

GADSdat	A GADSdat object.
vars	A character vector with the names of the text variables.
max_num	Maximum number of text variables. Additional text variables will be removed and NA codes given accordingly.
na_value	Which NA value should be given in cases of too many values on text variables.
na_label	Which value label should be given to the na_value.

Details

In some cases, multiple text variables contain the information of one variable (e.g. multiple answers to an open item). If this is a case, sometimes the number text variables displaying this variable should be limited. remove2NAchar allows shortening multiple character variables, this means character variables after max_num are removed from the GADSdat. Cases, which had valid responses on these removed variables are coded as missings (using na_value and na_label).

Value

Returns the modified GADSdat.

Examples

```
## create an example GADSdat
example_df <- data.frame(ID = 1:4,
                  citizenship1 = c("German", "English", "missing by design", "Chinese"),
                       citizenship2 = c(NA, "German", "missing by design", "Polish"),
                       citizenship3 = c(NA, NA, NA, "German"),
                       stringsAsFactors = FALSE)
gads <- import_DF(example_df)

## shorten character variables
gads2 <- remove2NAchar(gads, vars = c("citizenship1", "citizenship2", "citizenship3"),
                     na_value = -99, na_label = "missing: too many answers")
```

---

removeValLabels *Remove value labels.*

Description

Remove value labels of a variable as part of a GADSdat or all_GADSdat object.

Usage

```
removeValLabels(GADSdat, varName, value, valLabel = NULL)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of a variable name.
value	Numeric values.
valLabel	[optional] Regular expressions in the value labels corresponding to value.

Details

If the argument valLabel is provided the function checks for value and valLabel pairs in the meta data that match both arguments.

Value

Returns the GADSdat object with changed meta data.

Examples

```
# Remove a label based on value
extractMeta(pisa, "schtype")
pisa2 <- removeValLabels(pisa, varName = "schtype", value = 1)
extractMeta(pisa2, "schtype")

# Remove multiple labels based on value
extractMeta(pisa, "schtype")
pisa3 <- removeValLabels(pisa, varName = "schtype", value = 1:3)
extractMeta(pisa3, "schtype")

# Remove multiple labels based on value - valLabel combination
extractMeta(pisa, "schtype")
pisa4 <- removeValLabels(pisa, varName = "schtype",
                     value = 1:3, valLabel = c("Gymnasium", "other", "several courses"))
extractMeta(pisa4, "schtype")
```

---

reuseMeta *Use meta data for a variable from another* GADSdat.

Description

Transfer meta information from one GADSdat to another.

Usage

```
reuseMeta(
  GADSdat,
  varName,
  other_GADSdat,
  other_varName = NULL,
  missingLabels = NULL,
  addValueLabels = FALSE
)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Name of the variable that should get the new meta data.
other_GADSdat	GADSdat object imported via eatGADS including the desired meta information. Can also be a GADS db or an all_GADSdat object.
other_varName	Name of the variable that should get the new meta data in the other_GADSdat.
missingLabels	How should meta data for missing values be treated? If NULL, missing values are transferred as all other labels. If "drop", missing labels are dropped (useful for imputed data). If "leave", missing labels remain untouched.
addValueLabels	Should only value labels be added and all other meta information retained?

Details

Transfer of meta information can mean substituting the complete meta information, only adding value labels, or adding only "valid" missing labels. See the arguments missingLabels and addValueLabels for further information.

Value

Returns the original object with updated meta data.

Examples

```
# see createGADS vignette
```

---

splitGADS *Split* GADSdat *into hierarchy levels.*

Description

Split a GADSdat into multiple, specified hierarchical levels.

Usage

```
splitGADS(GADSdat, nameList)
```

Arguments

GADSdat	A GADSdat object.
nameList	A list of character vectors. The names in the list correspond the the hierarchy levels.

Details

The function takes a GADSdat object and splits it into its desired hierarchical levels (a all_GADSdat object). Hierarchy level of a variable is also accessible in the meta data via the column data_table. If not all variable names are included in the nameList, the missing variables will be dropped.

Value

Returns an all_GADSdat object, which consists of list with a list of all data frames "datList" and a single data frame containing all meta data information "allLabels". For more details see also mergeLabels.

Examples

```
# see createGADS vignette
```

---

stringAsNumeric *Transform string to numeric.*

Description

Transform a string variable within a GADSdat or all_GADSdat object to a numeric variable.

Usage

```
stringAsNumeric(GADSdat, varName)
```

Arguments

GADSdat	GADSdat object imported via eatGADS.
varName	Character string of a variable name.

Details

Applied to a GADSdat or all_GADSdat object, this function uses asNumericIfPossible to change the variable class and changes the format column in the meta data.

Value

Returns the GADSdat object with with the changed variable.

updateMeta *Update meta data.*

Description

Update the meta data of a GADSdat or all_GADSdat object according to the variables in a new data object.

Usage

```
updateMeta(GADSdat, newDat)
```

Arguments

GADSdat	GADSdat or all_GADSdat object.
newDat	data.frame or list of data.frames with the modified data. tibbles and data.tables are currently not supported and need to be transformed to data.frames beforehand.

Details

If the data of a GADSdat or a all_GADSdat has changed (supplied via newDat), updateMeta assimilates the corresponding meta data set. If variables have been removed, the corresponding meta data is also removed. If variables have been added, empty meta data is added for these variables. Factors are transformed to numerical and their levels added to the meta data set.

Value

Returns the original object with updated meta data (and removes factors from the data).

Examples

```
# see createGADS vignette
```

write_spss *Write a* GADSdat *object to a file*

Description

Write a GADSdat object, which contains meta information as value and variable labels to an SPSS file (sav) or Stata file (dta). See 'details' for some important limitations.

Usage

```
write_spss(GADSdat, filePath)

write_stata(GADSdat, filePath)
```

Arguments

GADSdat	A GADSdat object.
filePath	Path of sav file to write.

Details

The provided functionality relies on havens write_sav and write_dta functions.

Currently known limitations for write_spss are: (a) Missing codes for all character variables are dropped, (b) value labels for long character variables (> A10) are dropped, (c) under specific conditions very long character variables (> A254) are incorrectly displayed as multiple character variables in SPSS. Furthermore, write_spss currently does not support exporting date or time variables.

Currently known limitations for write_stata are: (a) Variable format is dropped, (b) missing codes are dropped.

Value

Writes file to disc, returns NULL.

Examples

```
# write to spss
tmp <- tempfile(fileext = ".sav")
write_spss(pisa, tmp)

# write to stata
tmp <- tempfile(fileext = ".dta")
write_stata(pisa, tmp)
```

---

write_spss2 *Write a* GADSdat *object to* txt *and* SPSS *syntax*

Description

Write a GADSdat object to a text file (txt) and an accompanying SPSS syntax file containing all meta information (e.g. value and variable labels).

Usage

```
write_spss2(GADSdat, filePath, syntaxPath)
```

Arguments

GADSdat	A GADSdat object.
filePath	Path of .txt file to write.
syntaxPath	Path of .sps file to write.

Details

This function is based on eatPreps writeSpss function and is currently under development.

Value

Writes sav file to disc, returns NULL.

Index