Package ‘eatGADS’

June 24, 2022

Title Data Management of Large Hierarchical Data

Version 0.20.0

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 (>= 0.5.0), haven (>= 2.4.0), plyr, eatTools (>= 0.4.0), tibble, data.table, hms, stats, utils, stringi

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], Karoline Sachse [ctb], Johanna Busse [ctb]

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

Repository CRAN

Date/Publication 2022-06-24 17:00:05 UTC

R topics documented:

applyChangeMeta .................................................. 3
applyLookup ..................................................... 4
applyLookup_expandVar ......................................... 5
applyNumCheck .................................................. 6
assimilateValLabels ............................................. 7
calculateScale .................................................. 8
R topics documented:

- `cbind.GADSdat` .................................................. 9
- `changeMissings` .................................................. 9
- `changeSPSSformat` .............................................. 10
- `changeValLabels` ................................................ 11
- `changeVarLabels` ................................................ 12
- `changeVarNames` ................................................ 13
- `check4SPSS` ..................................................... 13
- `checkEmptyValLabels` .......................................... 14
- `checkFormat` .................................................... 15
- `checkMissings` .................................................. 16
- `checkTrendStructure` .......................................... 17
- `checkUniqueness` ............................................... 18
- `checkUniqueness2` ............................................. 19
- `checkValue` ..................................................... 20
- `checkVarNames` ................................................ 21
- `clean_cache` ..................................................... 21
- `collapseColumns` ............................................... 22
- `collapseMC_Text` ............................................... 23
- `collapseMultiMC_Text` ....................................... 24
- `compareGADS` .................................................. 26
- `composeVar` .................................................... 27
- `createGADS` .................................................... 28
- `createLookup` .................................................. 29
- `createNumCheck` ............................................... 30
- `dummies2char` .................................................. 31
- `eatGADS` ........................................................ 32
- `equalGADS` ...................................................... 33
- `export_tibble` .................................................. 33
- `extractData` ..................................................... 34
- `extractDataOld` ................................................ 35
- `extractGADSdat` ............................................... 36
- `extractMeta` ..................................................... 37
- `extractVars` .................................................... 38
- `fac2dummies` ................................................... 39
- `fac2dummies_complex` ....................................... 39
- `fillImputations` ............................................... 40
- `fixEncoding` .................................................... 41
- `getChangeMeta` ................................................ 42
- `getGADS` ........................................................ 42
- `getGADS_fast` ................................................ 43
- `getTrendGADS` ................................................ 44
- `getTrendGADSOld` ............................................ 45
- `import_convertLabel` ......................................... 46
- `import_DF` ...................................................... 46
- `import_raw` ..................................................... 47
- `import_raw2` ................................................... 48
- `import_RDS` ..................................................... 49
- `import_spss` .................................................... 50
applyChangeMeta

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", "drop"),
  ...
)
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").

Furthermore, one might recode multiple old values in the same new value. This is currently only possible with existingMeta = "drop", which drops all related meta data on value level.

Value

Returns the modified GADSdat object.

Examples

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

```r
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

```r
## 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**

```r
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

```r
## create an example GADSdat
dataframeID = 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)
assimilateValLabels

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

---

assimilateValLabels  Assimilate value labels.

Description

Assimilate all value labels of multiple variables as part of a GADSdat or all_GADSdat object.

Usage

assimilateValLabels(GADSdat, varNames, lookup = NULL)

Arguments

GADSdat  GADSdat object imported via eatGADS.
varNames  Character string of a variable name.
lookup  Look up data.frame.

Details

Assimilation can be performed using all existing value labels or a look up table containing at least all existing value labels. Missing codes are reused based on the meta data of the first variable in varNames.

Value

Returns the GADSdat object with changed meta data and recoded values.
calculateScale

Calculate a scale.

Description

Calculate a scale variable based on multiple items.

Usage

```r
calculateScale(
  GADSdat, 
  items, 
  scale, 
  maxNA = length(items), 
  reportDescr = FALSE 
)
```

Arguments

- **GADSdat**: A data.frame or GADSdat object.
- **items**: A character vector with all item variable names.
- **scale**: A character vector with the scale name.
- **maxNA**: Maximum number of allowed NA values on the items.
- **reportDescr**: Should descriptive statistics be reported for the calculated scale.

Details

Descriptive statistics (including Cronbach’s alpha, credit to the psy package) are calculated and printed to the console. The new scale variable is automatically inserted right after the last item in the original GADSdat.

Value

Returns a GADSdat containing the newly computed variable.
Examples

```r
##
items <- paste0("norms.", letters[1:6])
pisa_new <- calculateScale(pisa, items = items, scale = "norms")
```

### `cbind.GADSdat`

**Bind two GADSdat objects into a single GADSdat object by columns.**

**Description**

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

**Usage**

```r
## S3 method for class 'GADSdat'
cbind(..., deparse.level = 1)
```

**Arguments**

- `...` Multiple `GADSdat` objects imported via `eatGADS`.
- `deparse.level` Argument is ignored in this method.

**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.

### `changeMissings`

**Change missing code.**

**Description**

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

**Usage**

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

Description

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

Usage

changeSPSSformat(GADSdat, varName, format)

Arguments

<table>
<thead>
<tr>
<th>GADSdat</th>
<th>GADSdat object imported via eatGADS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>varName</td>
<td>Character string of variable names.</td>
</tr>
<tr>
<td>format</td>
<td>A single string containing the new SPSS format, for example 'A25' or 'F10'.</td>
</tr>
</tbody>
</table>
**changeValLabels**

**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**

```r
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**

```r
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 <- changeVarLabels(pisa, varName = "repeated",
                        value = c(1, 2),
                        valLabel = c("no grade repetition", "grade repetition"))

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

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"))

check4SPSS

Check SPSS Compliance of Meta Data

Description

Function to check if variable names and labels, value labels and missing codes comply with SPSS requirements for meta data.

Usage

check4SPSS(GADSdat)
Arguments

GADSdat  GADSdat object imported via eatGADS.

Details

The function measures the length of variable names ("varNames_length", maximum of 64 characters) variable labels ("varLabels", maximum of 256 characters), value labels ("valLabels", maximum of 120 characters). Furthermore, missing codes are counted ("missings", maximum of three missing codes for character variables) and special characters are flagged in variable names ("varNames_special"). Check results are reported back on variable level, with the exception of "valLabels", which is a list with entries per violating variable.

Value

Returns a list with the entries "varNames_special", "varNames_length", "varLabels", "valLabels", and "missings".

Examples

# Change example data set (create a violating label)
pisa2 <- changeVarLabels(pisa, varName = "computer_age",
                         varLabel = paste(rep("3", 125), collapse = ""))

check4SPSS(pisa2)

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,
    output = c("list", "data.frame")
)

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

checkFormat

Description

Function to check if SPSS format statements are specified correctly in a GADSdat object.

Usage

checkFormat(GADSdat, type = "SPSS", changeFormat = TRUE)
Arguments

GADSdat

GADSdat object imported via eatGADS.

type

If type='other', the function nchar will be used to determine character lengths and decimals are not rounded to 16 decimal places. With type='SPSS' additional width for character variables will be added in order to let SPSS read in lengthy characters correctly and .

changeFormat

If changeFormat=TRUE the GADSdat meta data will be updated otherwise only information will be reported.

Details

The function compares SPSS format statements "format" and actual character length and decimal places of all variables in a GADSdat object and its meta data information. Mismatches are reported and can be automatically adjusted.

Value

Returns a GADSdat object.

Examples

# Change example meta information (create a value label with incorrect missing code)
pisa2 <- checkFormat(pisa)

---

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
)
checkTrendStructure

**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**

```r
# 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.

---

checkUniqueness  
*Check uniqueness of a variable.*

Description

Function to check if a variable is unique for all cases of an identifier variable.

Usage

checkUniqueness(GADSdat, varName, idVar)

Arguments

- **GADSdat**: GADSdat object imported via eatGADS.
- **varName**: Single string containing the variable name for which the check should be performed.
- **idVar**: Single string containing the identifier variable name.

Details

For example if missing values are multiple imputed and data is stored in a long format, checking the uniqueness of a variable within an identifier can be tricky. This function automates this task.

Value

Returns either TRUE if the variable is unique within each value for idVar or a GADSdat object including the not unique cases.

Examples

```r
## create an example GADSdat
iris2 <- iris
iris2$Species <- as.character(iris2$Species)
gads <- import_DF(iris2, checkVarNames = FALSE)

## check uniqueness
checkUniqueness(gads, varName = "Sepal.Length", idVar = "Species")
```
checkUniqueness2

Description

Function to check if a variable is unique for all cases of an identifier variable. This is a fast and more efficient version of checkUniqueness which always returns a logical of length one.

Usage

checkUniqueness2(GADSdat, varName, idVar, impVar)

Arguments

- **GADSdat**: GADSdat object imported via eatGADS.
- **varName**: Single string containing the variable name for which the check should be performed.
- **idVar**: Single string containing the name of the identifier variable.
- **impVar**: Single string containing the name of the imputation number.

Details

For example if missing values are multiple imputed and data is stored in a long format, checking the uniqueness of a variable within an identifier can be tricky. This function automates this task via reshaping the data into wide format and testing equality among the reshaped variables. Similar functionality (via matrices) is covered by lme4::isNested, which is more general and performs similarly.

Value

Returns a logical of length one.

Examples

```r
## create an example GADSdat
l <- 1000
long_df <- data.table::data.table(id = sort(rep(1:l, 15)),
                                  v1 = sort(rep(1:l, 15)),
                                  imp = rep(1:l, 15))
gads <- import_DF(long_df)
## check uniqueness
checkUniqueness2(gads, varName = "v1", idVar = "id", impVar = "imp")
```
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.
collapseColumns

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

```r
## (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**

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**

```r
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

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

g2 <- collapseMC_Text(g, 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

```r
collapseMultiMC_Text(  
  GADSdat,  
  mc_vars,  
  text_vars,  
  mc_var_4text,  
)  
```
\texttt{collapseMultiMC\_Text}

\begin{verbatim}
  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

\textbf{GADSdat} A GADSdat object.

\textbf{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 \texttt{matchValues\_varLabels} is recommended.

\textbf{text\_vars} A character vector with the names of the text variables which should be collapsed.

\textbf{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.

\textbf{var\_suffix} Variable suffix for the newly created GADSdat. If an empty character, the existing variables are overwritten.

\textbf{label\_suffix} Suffix added to variable label for the newly created or modified variables in the GADSdat.

\textbf{invalid\_miss\_code} Missing code which is given to new character variables if all text entries where recoded into the dichotomous variables.

\textbf{invalid\_miss\_label} Value label for invalid\_miss\_code.

\textbf{notext\_miss\_code} Missing code which is given to empty character variables.

\textbf{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. \texttt{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 (\texttt{createLookup} and \texttt{applyLookup\_expandVar}) and by matching the dummy variables to its underlying values stored in variable labels (\texttt{matchValues\_varLabels}).

The function recodes the dummy variables according to the character variables. Additionally, the \texttt{mc\_var\_4text} variable is recoded according to the final status of the \texttt{text\_vars} (exception: if the text variables were originally NA, \texttt{mc\_var\_4text} is left as it was).
Missing values in the character variables can be represented either by NAs 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 NAs on the character variables are given a specific missing code (notext_miss_code).

Value

Returns a GADSdat containing the newly computed variables.

Examples

```r
# 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,  # Old GADSdat
  GADSdat_new,  # New GADSdat
  varNames,     # Names of variables to compare
  output = c("list", "data.frame", "aggregated")
)
composeVar

Create a composite variable.

Description

Create a composite variable out of two variables.

Usage

composeVar(GADSdat, sourceVars, primarySourceVar, newVar)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GADSdat_old</td>
<td>GADSdat object imported via eatGADS.</td>
</tr>
<tr>
<td>GADSdat_new</td>
<td>GADSdat object imported via eatGADS.</td>
</tr>
<tr>
<td>varNames</td>
<td>Character string of variable names to be compared.</td>
</tr>
<tr>
<td>output</td>
<td>How should the output be structured?</td>
</tr>
</tbody>
</table>

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

```r
# 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")
```
Arguments

- **GADSdat**: GADSdat or all_GADSdat object imported via eatGADS.
- **sourceVars**: Character vector of length two containing the variable names which represent the sources of information.
- **primarySourceVar**: Character vector containing a single variable name. Which of the sourceVars should be preferred?
- **newVar**: Character vector containing the name of the new composite variable.

Details

A common use case for creating a composite variable is if there are multiple sources for the same information. For example, a child and the parents are asked about the child’s native language. In such cases a composite variable contains information from both variables, meaning that one source is preferred and the other source is used to substitute missing values.

Value

The modified GADSdat.

Examples

```r
# example data
dat <- data.frame(ID = 1:4,
nat_lang_child = c("Engl", "Ger", "missing", "missing"),
nat_lang_father = c("Engl", "Engl", "Ger", "missing"),
stringsAsFactors = TRUE)
gads <- import_DF(dat)
changeMissings(gads, "nat_lang_child", value = 3, missings = "miss")
changeMissings(gads, "nat_lang_father", value = 3, missings = "miss")

# compose variable
composeVar(gads, sourceVars = c("nat_lang_child", "nat_lang_father"),
            primarySourceVar = "nat_lang_child", newVar = "nat_lang_comp")
```

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

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

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

```r
# 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

```r
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.
createNumCheck

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

```r
# 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

```r
createNumCheck(GADSdat)
```

Arguments

- **GADSdat**: A GADSdat object.

Details

This function is currently under development.
Value

A data frame with the following variables:

<table>
<thead>
<tr>
<th>variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varLabel</td>
<td>Corresponding variable labels</td>
</tr>
<tr>
<td>min</td>
<td>Minimum value for the specific variable.</td>
</tr>
<tr>
<td>max</td>
<td>Maximum value for the specific variable.</td>
</tr>
<tr>
<td>value_new</td>
<td>Which value should be inserted if values exceed the specified range?</td>
</tr>
</tbody>
</table>

Examples

# tbd

dummies2char

Transform dummy variables to character variables.

Description

Convert a set of dummy variables into a set of character variables.

Usage

dummies2char(GADSdat, dummies, dummyValues, charNames)

Arguments

GADSdat   A GADSdat object.
dummies   A character vector with the names of the dummy variables.
dummyValues   A vector with the values which the dummy variables represent.
charNames   A character vector containing the new variable names.

Details

A set of dummy variables is transformed to an equal number of character variables. The character variables are aligned to the left and the remaining character variables are set to NA. For each new variable the missing codes of the respective dummy variable are reused.

Value

Returns a GADSdat.
Examples

```r
## create an example GADSdat
dummy_df <- data.frame(d1 = c("eng", "no eng", "eng"),
                       d2 = c("french", "french", "no french"),
                       d3 = c("no ger", "ger", "no ger"),
                       stringsAsFactors = TRUE)
dummy_g <- import_DF(dummy_df)

## transform dummy variables
dummy_g2 <- dummies2char(dummy_g, dummies = c("d1", "d2", "d3"),
                         dummyValues = c("english", "french", "german"),
                         charNames = c("char1", "char2", "char3"))
```

---

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`.
**equalGADS**

Test if Two GADSdat Objects are (Nearly) Equal

**Description**

Run tests to check whether two GADSdat objects are (nearly) equal. Variable names, number of rows in the data, meta data and data differences are checked and reported as a list output.

**Usage**

```r
equalGADS(
  target,
  current,
  metaExceptions = c("display_width", "labeled"),
  tolerance = sqrt(.Machine$double.eps)
)
```

**Arguments**

- `target`: A GADSdat object.
- `current`: A GADSdat object.
- `metaExceptions`: Should certain meta data columns be excluded from the comparison?
- `tolerance`: A numeric value greater than or equal to 0. Differences smaller than `tolerance` are not reported. The default value is close to 1.5e-8.

**Value**

Returns a list.

---

**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**

```r
export_tibble(GADSdat)
```

**Arguments**

- `GADSdat`: GADSdat object imported via eatGADS.
Details

This function is mainly intended for internal use.

Value

Returns a tibble.

Examples

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

---

**extractData**

**Extract Data**

Description

Extract data frame from a GADSdat object for analyses in R. Value labels can be selectively applied via defining `convertLabels` and `convertVariables`. For extracting meta data see `extractMeta`.

Usage

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

Arguments

- `GADSdat` A GADSdat object.
- `convertMiss` Should values tagged 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.
- `convertVariables` Character vector of variables names, which labels should be applied to. All other variables remain as numeric variables in the data. 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.
- `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`. 
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 on value level (missing conversion, value labels), so the data can be used for analyses in R. Variable labels are retained as label attributes on column level.

If factor are extracted via convertLabels == "factor", an attempt is made to preserve the underlying integers. 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, all others remain numeric
dat <- extractData(pisa, convertLabels = "factor", convertVariables = c("schtype", "ganztag"))

# convert only some variables to character, all others remain numeric
dat <- extractData(pisa, convertLabels = "factor", convertVariables = c("schtype", "ganztag"))
# schtype is now character
table(dat$schtype)
# schtype remains numeric
table(dat$gender)
extractGADSdat

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

- See `extractData` for the current functionality.

Value

- Returns a data frame.

---

**extractGADSdat**

Extract single GADSdat from all_GADSdat

Description

Function to extract a single GADSdat from an all_GADSdat object.

Usage

```r
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.
extractMeta

Value

Returns an GADSdat object.

Examples

# see createGADS vignette

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

```r
## create an example GADSdat
eexample_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"))
```
fac2dummies

Transform a factor variable to dummy variables.

Description
Convert a factor variable with n levels to n dummy variables.

Usage
fac2dummies(GADSdat, var)

Arguments
GADSdat A data.frame or GADSdat object.
var A character vector with the name of the factor variable.

Details
Newly created variables are named as the original variable with the suffix "_a", "_b" and so on. Variable labels are created by using the original variable label (if available) and adding the value label of the corresponding level. All missing codes are forwarded to all dummy variables.

Value
Returns a GADSdat containing the newly computed variables.

Examples
## create an example GADSdat
suppressMessages(gads <- import_DF(iris))

## transform factor variable
gads2 <- fac2dummies(gads, var = "Species")

fac2dummies_complex

Transform a complex factor variable to dummy variables.

Description
Convert a factor variable with complex factor levels (factor levels contain combinations of other factor levels) to dummy variables. Dummy variables are coded 1 ("yes") and 0 ("no").

Usage
fac2dummies_complex(GADSdat, var)
Arguments

GADSdat  A data.frame or GADSdat object.
var  A character vector with the name of the factor variable.

Details

The basic functionality of this function is analogous to fac2dummies. However, the function expects factor levels to only go to 9. Higher numbers are treated as combinations of factor levels, for example "13" as "1" and "3".

Value

Returns a GADSdat containing the newly computed variables.

Examples

```r
## create an example GADSdat
g_fac <- import_DF(df_fac)
g_fac <- recodeGADS(g_fac, varName = "fac", oldValues = c(1, 2, 3, 4, 5, 6), newValues = c(1, 12, 123, 2, 3, 23))
## transform factor variable
fac2dummies_complex(g_fac, "fac")
```

Description

Fill imputed values in a imputed GADSdat_imp object with original, not imputed values from a GADSdat.

Usage

```
fillImputations(GADSdat, GADSdat_imp, varName, varName_imp = varName, id, imp)
```

Arguments

GADSdat  A GADSdat object.
GADSdat_imp  A GADSdat object.
varName  A character vector of length 1 containing the variable name in GADSdat.
varName_imp  A character vector of length 1 containing the variable name in GADSdat_imp.
id  A character vector of length 1 containing the unique identifier column of both GADSdat.
imp  A character vector of length 1 containing the imputation number in GADSdat_imp.
Details

This function only fills in missing values in the imputed variable from the not imputed variable. It provides parts of the functionality of subImputations but does not check whether values have been mistakenly imputed. However, performance is increased substantially.

Value

The modified GADSdat_imp..

Examples

# tbd

---

fixEncoding  Remove special characters.

Description

Remove special characters from a character vector or a GADSdat object. Also suitable to fix encoding problems of a character vector or a GADSdat object, which was encoded presumably using UTF-8 and imported using ASCII encoding.

Usage

fixEncoding(x, input = c("other", "ASCII", "BRISE"))

Arguments

x  A character vector or GADSdat object.
input  Which encoding was used in import_spss.

Details

If entries are all upper case, special characters are also transformed to all upper case (e.g., "Æ" instead of "Æe").

Value

The modified character vector or GADSdat object.

Examples

fixEncoding(c("\U00C4pfel", "\U00C4PFEL", paste0("\U00DC", "ben"), paste0("\U00DC", "BEN")))
getChangeMeta  

Extract table for Meta Data Changes.

Description

Function to obtain a data frame from a GADSdat object 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)
getGADS_fast

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"))

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 multiple eatGADS data bases. 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**

```r
getTrendGADS(
  filePaths,
  vSelect = NULL,
  years,
  fast = TRUE,
  tempPath = tempdir(),
  verbose = TRUE
)
```

**Arguments**

- `filePaths`: Character vectors with paths to the eatGADS db files.
- `vSelect`: Variables from all GADS to be selected (as character vector).
- `years`: A numeric vector with identical length as `filePaths`.
- `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.
- `verbose`: Should the loading process be reported?

**Details**

This function extracts data from multiple GADS data bases. All data bases have to be created via `createGADS`. The data bases are joined via `rbind()` and a variable `year` is added, corresponding to the argument `years`. 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
getTrendGADSOld

Get data for trend reports.

Description

Support for linking error data bases has been removed from eatGADS. getGADSOld provides (for the time being) backwards compatibility, so linking errors can still be extracted automatically.

Usage

getTrendGADSOld(
  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

See getGADS for the current functionality.

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?
import_raw

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

```r
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

```r
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 varLabels has to contain exactly two variables, namely varName and varLabel. valLabels 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.
import_raw2

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"),
value = c(1, 2, 3),
stringsAsFactors = FALSE)
valLabels <- data.frame(varName = c("grade", "grade", "grade"),
value = c(1, 2, 3),
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.
**import_RDS**

**Value**

Returns a GADSdat object.

**Examples**

```r
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)
```

---

**Description**

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

**Usage**

```r
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`.
import_spss

Value

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

Description

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

Usage

import_spss(
  filePath,
  checkVarNames = TRUE,
  labeledStrings = c("drop", "keep", "transform"),
  encoding = NULL
)

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? If "drop" (default), all labeled strings are dropped and NAs occur in the meta data. If "transform", all underlying values are transformed to numeric. If "keep", value labels stay untouched. However, the latter possibly corrupts all labeled values.
encoding         The character encoding used for the file. The default, NULL, use the encoding specified in the file, but sometimes this value is incorrect and it is useful to be able to override it.

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.

In some special cases, .sav files seem to consist of a mix of different encoding types. In such cases, haven might throw an error if the encoding argument is not specified or UTF-8 is selected as encoding. To circumvent this problem we recommend using encoding = "ASCII" and fixing the resulting issues manually. For example, fixEncoding provides some fixes for encoding issues specific to the German language.
import_stata

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.
inspectDifferences  

Inspect differences in a variable.

Description

Inspect differences between two GADSdat objects in a specific variable.

Usage

inspectDifferences(varName, GADSdat1, GADSdat2, id)

Arguments

varName  
A character vector of length 1 containing the variable name.

GADSdat1  
A GADSdat object.

GADSdat2  
A GADSdat object.

id  
A character vector of length 1 containing the unique identifier column of both GADSdat.

Details

Two GADSdat objects can be compared using equalGADS. If differences in the data are indicated for specific variables, these variables can be further inspected using inspectDifferences.

Value

A list.

Examples

# create a second GADS with different data
pisa2 <- pisa
pisa2$dat$age[400:nrow(pisa$dat)] <- sample(pisa2$dat$age[400:nrow(pisa$dat)])

# inspect via equalGADS()
equalGADS(pisa, pisa2)

# inspect via inspectDifferences()
ispectDifferences("age", GADSdat1 = pisa, GADSdat2 = pisa2, id = "idstud")
inspectMetaDifferences

Inspect differences in a variable.

**Description**

Inspect differences between two GADSdat objects in a specific variable.

**Usage**

```r
inspectMetaDifferences(varName, GADSdat1, GADSdat2)
```

**Arguments**

- `varName`: A character vector of length 1 containing the variable name.
- `GADSdat1`: A GADSdat object.
- `GADSdat2`: A GADSdat object.

**Details**

Two GADSdat objects can be compared using `equalGADS`. If differences in the meta data are indicated for specific variables, these variables can be further inspected using `inspectMetaDifferences`.

**Value**

A list.

**Examples**

```r
# create a second GADS with different meta data
pisa2 <- pisa
pisa2 <- changeVarLabels(pisa2, varName = "sameteach", varLabel = "Same math teacher")
pisa2 <- recodeGADS(pisa2, varName = "sameteach", oldValues = c(1, 2), newValues = c(0, 1))

# inspect via equalGADS()
equalGADS(pisa, pisa2)

# inspect via inspectMetaDifferences()
inspectMetaDifferences("sameteach", GADSdat1 = pisa, GADSdat2 = pisa2)
```
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**

```r
# 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))`
merge.GADSdat

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

```r
# 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

```r
## 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

```r
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.
miss2NA

Value

Returns an all_GADS dat 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 GADS dat 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GADSdat</td>
<td>A data.frame or GADSdat object.</td>
</tr>
<tr>
<td>vars</td>
<td>A character vector with all variables that should be transformed to factor.</td>
</tr>
<tr>
<td>var_suffix</td>
<td>Variable suffix for the newly created GADSdat. If an empty character, the existing variables are overwritten.</td>
</tr>
<tr>
<td>label_suffix</td>
<td>Suffix added to variable label for the newly created variable in the GADSdat.</td>
</tr>
</tbody>
</table>

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

```r
## create an example GADSdat
data.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(data.example_df)
## transform multiple strings
gads2 <- multiChar2fac(gads, vars = c("citizenship1", "citizenship2"))
## set values to missings
gads3 <- checkMissings(gads2, missingLabel = "missing")
```

Description

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

Usage

```r
namesGADS(GADS)
```
orderLike

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)

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**


---

**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 NAs. recodeString2NA can be used to recode all empty strings to NA beforehand.

Value

Returns the recoded GADSdat.

Examples

```r
# 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 variable.

Description

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

Usage

```r
recodeGADS(
  GADSdat,
  varName,
  oldValues,
  newValues,
  existingMeta = c("stop", "value", "value_new", "drop")
)
```
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. Beyond that, unlabeled variables and values are recoded as well. 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").

Furthermore, one might recode multiple old values in the same new value. This is currently only possible with existingMeta = "drop", which drops all related meta data on value level.

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.
removeValLabels

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
```r
## create an example GADSdat
dexample_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.
reuseMeta

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")

# Remove multiple labels based on value - valLabel combination
extractMeta(pisa, "schtype")
pisa5 <- removeValLabels(pisa, varName = "schtype",
value = 1:3, valLabel = c("Gymnasium", "other", "several courses"))
extractMeta(pisa5, "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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GADSdat</td>
<td>GADSdat object imported via eatGADS.</td>
</tr>
<tr>
<td>varName</td>
<td>Name of the variable that should get the new meta data.</td>
</tr>
<tr>
<td>other_GADSdat</td>
<td>GADSdat object imported via eatGADS including the desired meta information.</td>
</tr>
<tr>
<td>other_varName</td>
<td>Can also be a GADS db or an all_GADSdat object.</td>
</tr>
<tr>
<td>missingLabels</td>
<td>Name of the variable that should get the new meta data in the other_GADSdat.</td>
</tr>
<tr>
<td>addValueLabels</td>
<td>Name of the variable that should get the new meta data in the other_GADSdat.</td>
</tr>
</tbody>
</table>

Value

Returns the GADSdat object with changed meta data.
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. If "only", all valid value labels are dropped.

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, adding only "valid" or adding only "miss" 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.
stringAsNumeric

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.

---

subImputations    Substitute imputed values.

---

Description

Substitute imputed values in a imputed GADSdat_imp object with original, not imputed values from a GADSdat.

Usage

subImputations(GADSdat, GADSdat_imp, varName, varName_imp = varName, id, imp)
Arguments

- **GADSdat**: A GADSdat object.
- **GADSdat_imp**: A GADSdat object.
- **varName**: A character vector of length 1 containing the variable name in GADSdat.
- **varName_imp**: A character vector of length 1 containing the variable name in GADSdat_imp.
- **id**: A character vector of length 1 containing the unique identifier column of both GADSdat.
- **imp**: A character vector of length 1 containing the imputation number in GADSdat_imp.

Details

There are two cases in which values are substituted: (a) there are missings in varName_imp, (b) values have been imputed even though there is valid information in varName.

Value

The modified GADSdat_imp.

Examples

```r
# tbd
```

---

**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

```r
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

```r
write_spss(GADSdat, filePath)
```

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) value labels for long character variables (> A10) are dropped, (b) 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

```r
# 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

```r
write_spss2(
  GADSdat,
  txtPath,
  spsPath = NULL,
  savPath = NULL,
  dec = ",",
  fileEncoding = "UTF-8",
  chkFormat = TRUE,
  ...
)
```

#### Arguments

- **GADSdat**: A GADSdat object.
- **txtPath**: Path of .txt file to write, including file name and ending .txt. No default.
- **spsPath**: Path of .sps file to write, including file name and ending .sps. Default Path is txtPath.
- **savPath**: Path of .sav file to write, including file name and ending .sav. Default Path is spsPath.
- **dec**: Decimal delimiter for your SPSS version. Other values for dec than "," or "." are not implemented yet.
- **fileEncoding**: Data file encoding for SPSS. Default is "UTF-8".
- **chkFormat**: Whether format checks via checkFormat should be performed.
- **...**: Arguments to pass to checkFormat in particular changeFormat=FALSE if needed.
**write_spss2**

**Details**

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

**Value**

Writes a `.txt` and an `.sav` file to disc, returns nothing.

**Examples**

```r
# write to spss
tmp_txt <- tempfile(fileext = " .txt")
write_spss2(pisa, txtPath = tmp_txt)
```
Index

* datasets
  pisa, 60

applyChangeMeta, 3, 10–13, 62
applyLookup, 4, 5, 6, 22, 29
applyLookup_expandVar, 5, 5, 25, 29
applyNumCheck, 6
asNumericIfPossible, 67
assimilateValLabels, 7
calculateScale, 8
cbind.GADSdat, 9
changeMissings, 9, 58
changeSPSSformat, 10
changeValLabels, 11, 62
changeVarLabels, 12
changeVarNames, 13
check4SPSS, 13
checkEmptyValLabels, 14
checkFormat, 15
checkMissings, 16, 50, 58
checkMissingValLabels (checkEmptyValLabels), 14
checkTrendStructure, 17
checkUniqueness, 18, 19
checkUniqueness2, 19
checkValue, 20
checkVarNames, 21
clean_cache, 21
collapseColumns, 4, 22, 29
collapseMC_Text, 23
collapseMultiMC_Text, 24, 55
compareGADS, 26
composeVar, 27
createDB, 29, 32, 43, 44
createGADS, 28, 32, 44, 56
createLookup, 4–6, 22, 25, 29, 61, 62
createNumCheck, 6, 7, 30
dbPull, 43, 44
dummies2char, 31
eatGADS, 32
equalGADS, 33, 52, 53
export_tibble, 33
extractData, 32, 34, 36, 44
extractDataOld, 35
extractGADSdat, 36
extractMeta, 32, 34, 37, 44
extractVars, 38
fac2dummies, 39, 40
fac2dummies_complex, 39
fillImputations, 40
fixEncoding, 41, 50
getChangeMeta, 3, 4, 10–13, 42, 62
getGADS, 32, 42, 44, 45
getGADS_fast, 22, 43, 44, 45
getTrendGADS, 22, 32, 44
getTrendGADSOld, 45
import_convertLabel, 46
import_DF, 32, 46, 49
import_raw, 47, 48
import_raw2, 48
import_RDS, 49
import_spss, 32, 41, 46–48, 50, 57
import_stata, 51
inspectDifferences, 52
inspectMetaDifferences, 53
labelsGADS, 54
matchValues_varLabels, 25, 54
merge_GADSdat, 55
mergeLabels, 29, 32, 36, 56, 66
miss2NA, 57
multiChar2fac, 57
namesGADS, 32, 58
INDEX

orderLike, 59
pisa, 60
read_spss, 33
recode2NA, 60, 62, 63
recodeGADS, 26, 61
recodeString2NA, 63
remove2NAchar, 63
removeValLabels, 64
removeVars(extractVars), 38
reuseMeta, 65
splitGADS, 66
sqlite_keywords, 21
stringAsNumeric, 67
subImputations, 67
updateMeta, 38, 68
write_dta, 69
write_sav, 69
write_spss, 69
write_spss2, 70
write_stata(write_spss), 69