Package 'dataMaid'

July 28, 2019

Type Package

Title A Suite of Checks for Identification of Potential Errors in a Data Frame as Part of the Data Screening Process

Version 1.3.2

Date 2019-07-26

Description Data screening is an important first step of any statistical analysis. dataMaid auto generates a customizable data report with a thorough summary of the checks and the results that a human can use to identify possible errors. It provides an extendable suite of test for common potential errors in a dataset.

URL `https://github.com/ekstroem/dataMaid`, `https://doi.org/10.18637/jss.v090.i06`

BugReports `https://github.com/ekstroem/dataMaid/issues`

Imports ggplot2, gridExtra, haven, htmltools, magrittr, methods, pander, robustbase (>= 0.93-2), stringi, whoami

Suggests knitr, rmarkdown (>= 1.10), testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData true

ByteCompile true

License GPL-2

RoxygenNote 6.1.1

Collate 'aggregateForBarplot.R' 'aggregateForHistogram.R' 'allCheckFunctions.R' 'allClasses.R' 'allSummaryFunctions.R' 'allVisualFunctions.R' 'allXFunctions.R' 'makeXFunction.R' 'visualFunction.R' 'basicVisual.R' 'summaryFunction.R' 'centralValue.R' 'check.R' 'checkResult.R' 'messageGenerator.R' 'checkFunction.R' 'identifyMissing.R' 'minMax.R' 'classes.R' 'clean.R' 'countMissing.R' 'dataMaid-package.R' 'dataMaid_as_factor.R' 'description.R' 'identifyCaseIssues.R'

'identifyLoners.R' 'identifyNums.R' 'identifyOutliers.R'
'identifyOutliersTBStyle.R' 'identifyWhitespace.R' 'isCPR.R'
'isSingular.R' 'isEmpty.R' 'isKey.R' 'isSupported.R'
'makeCodebook.R' 'makeDataReport.R' 'misc.R' 'quartiles.R'
'refCat.R' 'render.R' 'setChecks.R' 'setSummaries.R'
'setVisuals.R' 'smartNum.R' 'standardVisual.R' 'summarize.R'
'summaryResult.R' 'tableVisual.R' 'uniqueValues.R'
'unpackLabelled.R' 'variableType.R' 'visualize.R'

NeedsCompilation no

Author Anne Helby Petersen [aut],
Claus Thorn Ekstrøm [aut, cre]

Maintainer Claus Thorn Ekstrøm <ekstrom@sund.ku.dk>

Repository CRAN

Date/Publication 2019-07-27 22:00:06 UTC

R topics documented:

`allCheckFunctions` *Overview of all available checkFunctions*

Description

Produce an overview of all functions of class `checkFunction` available in the workspace or imported from packages. This overview includes the descriptions and a list of what classes the functions are each intended to be called on.

Usage

```
allCheckFunctions()
```

Value

An object of class `functionSummary`. This object has entries `$name` (the function names), `$description` (the function descriptions, as obtained from their `description` attributes) and `$classes` (the classes each function is indeded to be called on, as obtained from their `classes` attributes).

See Also

`checkFunction` `allVisualFunctions` `allSummaryFunctions`

Examples

```
allCheckFunctions()
```

`allClasses` *Vector of all variable classes in* `dataMaid`

Description

Returns the names of the eight data classes for which `dataMaid` is implemented, namely `"character"`, `"Date"`, `"factor"`, `"integer"`, `"labelled"`, `"haven_labelled"`, `"logical"` and `"numeric"`.

Usage

```
allClasses()
```

Examples

```
allClasses()
```

allSummaryFunctions *Overview of all available summaryFunctions*

Description

Produce an overview of all functions of class `summaryFunction` available in the workspace or imported from packages. This overview includes the descriptions and a list of what classes the functions are each intended to be called on.

Usage

```
allSummaryFunctions()
```

Value

An object of class `functionSummary`. This object has entries `$name` (the function names), `$description` (the function descriptions, as obtained from their `description` attributes) and `$classes` (the classes each function is indeded to be called on, as obtained from their `classes` attributes).

See Also

`summaryFunction` `allVisualFunctions` `allCheckFunctions`

Examples

```
allSummaryFunctions()
```

allVisualFunctions *Overview of all available visualFunctions*

Description

Produce an overview of all functions of class `visualFunction` available in the workspace or imported from packages. This overview includes the descriptions and a list of what classes the functions are each intended to be called on.

Usage

```
allVisualFunctions()
```

Value

An object of class `functionSummary`. This object has entries `$name` (the function names), `$description` (the function descriptions, as obtained from their `description` attributes) and `$classes` (the classes each function is indeded to be called on, as obtained from their `classes` attributes).

See Also

`visualFunction` `allCheckFunctions` `allSummaryFunctions`

Examples

```
allVisualFunctions()
```

---

artData *Semi-artificial data about masterpieces of art*

Description

A dataset with information about 200 painting and their painters. Each observation in the dataset corresponds to a painting. A single artificial variable, namely an artist ID variable, has been included. Otherwise the information should be truthful.

Usage

```
artData
```

Format

A data frame with 200 rows and 11 variables.

ArtistID A unique ID used for cataloging the artists (fictional).

ArtistName The name of the artist.

NoOfMiddlenames The number of middlenames the artist has.

Title The title of the painting.

Year The approximate year in which the painting was made.

Location The current location of the painting.

Continent The continent of the current location of the painting.

Width The width of the painting, in centimeters.

Height The height of the painting, in centimers.

Media The media/materials of the painting.

Movement The artistic movement(s) the painting belongs to.

Source

Semi-artificial dataset constructed based on the Master Works of Art dataset available from Data Explorer[1].

Examples

```
data(artData)
```

`basicVisual` *Produce distribution plots in the base R (graphics) style using* `plot` *and* `barplot`

Description

Plot the distribution of a variable, depending on its data class, using the base R plotting functions. Note that `basicVisual` is a `visualFunction`, compatible with the `visualize` and `makeDataReport` functions.

Usage

```
basicVisual(v, vnam, doEval = TRUE)
```

Arguments

`v`	The variable (vector) to be plotted.
`vnam`	The name of the variable which will appear as the title of the plot.
`doEval`	If TRUE, the plot itself is returned. Otherwise, the function returns a character string containing standalone R code for producing the plot.

Details

For character, factor, logical and (haven_)labelled variables, a barplot is produced. For numeric, integer or Date variables, `basicVisual` produces a histogram instead. Note that for integer and numeric variables, all non-finite (i.e. `NA`, `NaN`, `Inf`) values are removed prior to plotting. For character, factor, (haven_)labelled and logical variables, only `NA` values are removed.

See Also

`visualize`, `standardVisual`

[1] `http://www.data-explorer.com/data`

Examples

```
## Not run:
#Save a variable
  myVar <- c(1:10)
#Plot a variable
  basicVisual(myVar, "MyVar")

#Produce code for plotting a variable
  basicVisual(myVar, "MyVar", doEval = FALSE)

## End(Not run)
```

---

basicVisualCFLB *importFrom stats na.omit*

Description

importFrom stats na.omit

Usage

```
basicVisualCFLB(v, vnam, doEval = TRUE)
```

Arguments

v	The variable (vector) to be plotted.
vnam	The name of the variable which will appear as the title of the plot.
doEval	If TRUE, the plot itself is returned. Otherwise, the function returns a character string containing standalone R code for producing the plot.

---

bigPresidentData *Semi-artificial data about the US presidents (extended version)*

Description

A dataset with information about the first 45 US presidents as well as a 46th person, who is not a US president, and a duplicate of one of the 45 actual presidents. The dataset was constructed to show the capabilities of `dataMaid` and therefore, it has been constructed to include errors and miscodings. Each observation in the dataset corresponds to a person. The dataset uses the non-standard class `Name` which is simply an attribute that has been added to two variables in order to show how `dataMaid` handles non-supported classes. Note that the dataset is an extended and more error-filled version of the dataset `presidentData` which is also included in the package.

Usage

```
bigPresidentData
```

Format

A data frame with 47 rows and 15 variables.

lastName A `Name` type variable containing the last name of the president.

firstName A `Name` type variable containing the first name of the president.

orderOfPresidency A factor variable indicating the order of the presidents (with George Washington as number 1 and Donald Trump as number 45).

birthday A Date variable with the birthday of the president.

dateOfDeath A Date variable with the date of the president's death.

stateOfBirth A character variable with the state in which the president was born.

party A charcter variable with the party to which the president was associated.

presidencyBeginDate A Date variable with the date of inauguration of the president.

presidencyEndDate A Date variable with the date at which the presidency ends.

assassinationAttempt A numeric variable indicating whether there was an assassination attempt (`1`) or not (`0`) on the president.

sex A factor variable with the sex of the president.

ethnicity A factor variable with the ethnicity of the president.

presidencyYears A numeric variable with the duration of the presidency, in years.

ageAtInauguration A character variable with the age at inauguration.

favoriteNumber A `complex` type variable with a fictional favorite number for each president.

Source

Artificial dataset constructed based on the US president dataset available from Data Explorer[2].

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (`https://doi.org/10.18637/jss.v090.i06`).

Examples

```
data(bigPresidentData)
```

[2] `http://www.data-explorer.com/data`

`centralValue` *summaryFunction for central values*

Description

A `summaryFunction`, intended to be called from `summarize`, which returns the central value of a variable. For numeric and integer variables, this is the median. For character, factor, (have_)labelled, Date and logical variables, the central value is the mode (i.e. the value that occurs the largest number of times).

Usage

```
centralValue(v, ...)
```

Arguments

`v`	A variable (vector).
`...`	Extra arguments to be passed to class-specific functions. These incluse `maxDecimals` (default is 2) which controls the rounding of integer and numeric values.

Details

Note that NA, NaN and Inf values are ignored for numeric and integer variables, while only NA values are ignored for factor, character, Date and (haven_)labelled variables. No values are ignored for logical variables.

Value

An object of class `summaryResult` with the following entries: `$feature` (the mode/median),`$result` (the central value of `v`) and `$value` (identical to `$result`).

If the mode is returned and it is not uniquely determined, the first value qualifying as a mode is returned, when the variable is sorted according to `sort`.

See Also

`summaryFunction`, `summarize`, `summaryResult`, `allSummaryFunctions`

Examples

```
#central value of an integer variable:
  centralValue(c(rep(1, 25), rep(2, 10), rep(3, 20)))

#central value of a character variable:
  centralValue(as.character(c(rep(1, 20), rep(2, 10), rep(3, 20))))
```

check *Perform checks of potential errors in variable/dataset*

Description

Run a set of validation checks to check a variable vector or a full dataset for potential errors. Which checks are performed depends on the class of the variable and on user inputs.

Usage

```
check(v, nMax = 10, checks = setChecks(), ...)
```

Arguments

v	the vector or the dataset (data.frame) to be checked.
nMax	If a check is supposed to identify problematic values, this argument controls if all of these should be pasted onto the outputted message, or if only the first nMax should be included. If set to Inf, all problematic values are printed.
checks	A list of checks to use on each supported variable type. We recommend using setChecks for creating this list and refer to the documentation of this function for more details.
...	Other arguments that are passed on to the checking functions. These includes general parameters controlling how the check results are formatted (e.g. maxDecimals, which controls the number of decimals printed for numerical, problematic values).

Details

It should be noted that the default options for each variable type are returned by calling e.g. defaultCharacterChecks(), defaultFactorChecks(), defaultNumericChecks(), etc. A complete overview of all default options can be obtained by calling setChecks(). Moreover, all available checkFunctions (including both locally defined functions and functions imported from dataMaid or other packages) can be viewed by calling allCheckFunctions().

Value

If v is a variable, a list of objects of class checkResult, which each summarizes the result of a checkFunction call performed on v. See checkResult for more details. If V is a data.frame, a list of lists of the form above is returned instead with one entry for each variable in v.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (https://doi.org/10.18637/jss.v090.i06).

See Also

setChecks, allCheckFunctions checkResult checkFunction, defaultCharacterChecks, defaultFactorChecks, defaultLabelledChecks, defaultHavenlabelledChecks, defaultNumericChecks, defaultIntegerChecks, defaultLogicalChecks, defaultDateChecks

Examples

```
x <- 1:5
check(x)

#Annoyingly coded missing as 99
y <- c(rnorm(100), rep(99, 10))
check(y)

#Check y for outliers and print 4 decimals for problematic variables
check(y, checks = setChecks(numeric = "identifyOutliers"), maxDecimals = 4)

#Change what checks are performed on a variable, now only identifyMissing is called
# for numeric variables
check(y, checks = setChecks(numeric = "identifyMissing"))

#Check a full data.frame at once
data(cars)
check(cars)

#Check a full data.frame at once, while changing the standard settings for
#several data classes at once. Here, we ommit the check of miscoded missing values for factc
#and we only do this check for numeric variables:
check(cars, checks = setChecks(factor = defaultFactorChecks(remove = "identifyMissing"),
  numeric = "identifyMissing"))
```

---

checkFunction *Create an object of class checkFunction*

Description

Convert a function, f, into an S3 checkFunction object. This adds f to the overview list returned by an allCheckFunctions() call.

Usage

```
checkFunction(f, description = NULL, classes = NULL)
```

Arguments

f	A function. See details and examples below for the exact requirements of this function.
description	A character string describing the check performed by f. If NULL (the default), the name of f will be used instead.
classes	The classes for which f is intended to be called. If NULL (the default), one of two things happens. If f is not a S3 generic function, the classes attribute of f will be an empty character string. If f is a S3 generic function, an automatic look-up for methods will be conducted, and the classes attribute will then be filled out automatically. Note that the function allClasses (listing all classes used in dataMaid) might be useful.

Details

checkFunction represents the functions used in check and makeDataReport for performing error checks and quality control on variables in dataset.

An example of defining a new checkFunction is given below. Note that the minimal requirements for such a function (in order for it to be compatible with check() and makeDataReport()) is the following input/output-structure: It must input at least two arguments, namely v (a vector variable) and .... Additional implemented arguments from check() and makeDataReport() include nMax and maxDecimals, see e.g. the pre-defined checkFunction identifyMissing for more details about how these arguments should be used. The output must be a list with at least the two entries $problem (a logical indicating whether a problem was found) and $message (a character string message describing the problem). However, if the result of a checkFunction is furthermore appended with a $problemValues entry (including the values from the variable that caused the problem, if relevant) and converted to a checkResult object, a print() method also becomes available for consistent formatting of checkFunction results.

Note that all available checkFunctions are listed by the call allCheckFunctions() and we recommed looking into these function, if more knowledge about checkFunctions is required.

Value

A function of class checkFunction which has to attributes, namely classes and description.

See Also

allCheckFunctions, check, makeDataReport, messageGenerator, checkResult

Examples

```
#Define a minimal requirement checkFunction that can be called
#from check() and makeDataReport(). This function checks whether all
#values in a variable are of equal length and that this
#length is then also larger than 10:
isID <- function(v, nMax = NULL, ...) {
  out <- list(problem = FALSE, message = "")
```

```
  if (class(v) %in% c("character", "factor", "labelled", "haven_labelled", "numeric", "integ
    v <- as.character(v)
    lengths <- nchar(v)
    if (all(lengths > 10) & length(unique(lengths)) == 1) {
      out$problem <- TRUE
      out$message <- "Warning: This variable seems to contain ID codes!"
     }
   }
  out
}

#Convert it into a checkFunction
isID <- checkFunction(isID, description = "Identify ID variables (long, equal length values)
  classes = allClasses())

#Call isID
isID(c("12345678901", "23456789012", "34567890123", "45678901234"))

#isID now appears in a allCheckFunctions() call:
allCheckFunctions()


#Define a new checkFunction using messageGenerator() for generating
#the message and checkResult() for getting a printing method
#for its output. This function identifies values in a variable
#that include a colon, surrounded by alphanumeric characters. If
#at least one such value is found, the variable is flagged as
#having a problem:
identifyColons <- function(v, nMax = Inf, ... ) {
 v <- unique(na.omit(v))
 problemMessage <- "Note: The following values include colons:"
 problem <- FALSE
 problemValues <- NULL
 problemValues <- v[sapply(gregexpr("[[:xdigit:]]:[[:xdigit:]]", v),
                           function(x) all(x != -1))]
 if (length(problemValues) > 0) {
   problem <- TRUE
 }
 problemStatus <- list(problem = problem,
                       problemValues = problemValues)
 outMessage <- messageGenerator(problemStatus, problemMessage, nMax)
 checkResult(list(problem = problem,
                  message = outMessage,
                  problemValues = problemValues))
}

#Make it a checkFunction:
identifyColons <- checkFunction(identifyColons,
     description = "Identify non-suffixed nor -prefixed colons",
     classes = c("character", "factor", "labelled", "haven_labelled"))

#Call it:
identifyColons(1:100)
```

```
identifyColons(c("a:b", 1:10, ":b", "a:b:c:d"))

#identifyColons now appears in a allCheckFunctions() call:
allCheckFunctions()

#Define a checkFunction that looks for negative values in numeric
#or integer variables:
identifyNeg <- function(v, nMax = Inf, maxDecimals = 2, ...) {
  problem <- FALSE
  problemValues <- printProblemValues <- NULL
  problemMessage <- "Note: The following negative values were found:"
  negOcc <- unique(v[v < 0])
  if (length(negOcc > 0)) {
    problemValues <- negOcc
    printProblemValues <- round(negOcc, maxDecimals)
    problem <- TRUE
  }
  outMessage <- messageGenerator(list(problem = problem,
    problemValues = printProblemValues), problemMessage, nMax)
  checkResult(list(problem = problem,
                   message = outMessage,
                   problemValues = problemValues))
}

#Make it a checkFunction
identifyNeg <- checkFunction(identifyNeg, "Identify negative values",
  classes = c("integer", "numeric"))

#Call it:
identifyNeg(c(0:100))
identifyNeg(c(-20.1232323:20), nMax = 3, maxDecimals = 4)

#identifyNeg now appears in a allCheckFunctions() call:
 allCheckFunctions()
```

`checkResult` *Create object of class checkResult*

Description

Convert a list resulting from the checks performed in a `checkFunction` into a `checkResult` object, thereby supplying it with a `print()` method.

Usage

```
checkResult(ls)
```

Arguments

ls	A list with entries $problem (logical indicating whether a problem was found), $message (a character string containing a message describing the problem) and $problemValues (the values in the checked variables that were marked as problematic). Note that $message and $problemValues can be left empty (i.e. "" and NULL, respectively), if they are not relevant.

Value

A S3 object of class checkResult, identical to the inputted list, ls, except for its class attribute.

See Also

checkFunction

---

classes *Extract the contents of the attribute* classes

Description

If the object, x, is itself of class checkFunction, summaryFunction or visualFunction, the contents of x's attribute classes is returned. Otherwise, NULL is returned.

Usage

```
classes(x)

classes(x) <- value
```

Arguments

x	The object for which the classes attribute should be extracted.
value	New value

Value

The classes for which x is intended to be called, given as a vector of characters.

Examples

```
#Extract the classes of the checkFunction identifyMissing
classes(identifyMissing)

#Extract the classes of the summaryFunction minMax
classes(minMax)

#Extract the classes of the visualFunction basicVisual
classes(basicVisual)
```

clean *Produce a data cleaning overview document (deprecated version)*

Description

NOTE: This function has been replaced by `makeDataReport`. The current function is no longer updated and it is only included for backwards compatability.

Usage

```
clean(data, output = c("pdf", "html"), render = TRUE, useVar = NULL,
  ordering = c("asIs", "alphabetical"), onlyProblematic = FALSE,
  labelled_as = c("factor"), mode = c("summarize", "visualize",
  "check"), smartNum = TRUE, preChecks = c("isKey", "isSingular",
  "isSupported"), file = NULL, replace = FALSE, vol = "",
  standAlone = TRUE, twoCol = TRUE, quiet = TRUE,
  openResult = TRUE, characterChecks = defaultCharacterChecks(),
  factorChecks = defaultFactorChecks(),
  labelledChecks = defaultLabelledChecks(),
  numericChecks = defaultNumericChecks(),
  integerChecks = defaultIntegerChecks(),
  logicalChecks = defaultLogicalChecks(),
  dateChecks = defaultDateChecks(), allChecks = NULL,
  characterSummaries = defaultCharacterSummaries(),
  factorSummaries = defaultFactorSummaries(),
  labelledSummaries = defaultLabelledSummaries(),
  numericSummaries = defaultNumericSummaries(),
  integerSummaries = defaultIntegerSummaries(),
  logicalSummaries = defaultLogicalSummaries(),
  dateSummaries = defaultDateSummaries(), allSummaries = NULL,
  allVisuals = "standardVisual", listChecks = TRUE, maxProbVals = 10,
  maxDecimals = 2, addSummaryTable = TRUE, reportTitle = NULL,
  treatXasY = NULL, ...)
```

Arguments

`data`	The dataset to be checked. This dataset should be of class `data.frame`, `tibble` or `matrix`. If it is of classs `matrix`, it will be converted to a `data.frame`.
`output`	Output format. Options are `"pdf"` (the default), and `"html"`
`render`	Should the output file be rendered (defaults to `TRUE`), i.e. should a pdf/html document be generated and saved to the disc?
`useVar`	Variables to clean. If `NULL` (the default), all variables in `data` are included. If a vector of variable names is supplied, only the variables in `data` that are also in `useVar` are included in the data cleaning overview document.

`ordering` Choose the ordering of the variables in the variable presentation. The options are "asIs" (ordering as in the dataset) and "alphabetical" (alphabetical order).

`onlyProblematic` A logical. If `TRUE`, only the variables flagged as problematic in the check step will be included in the variable list.

`labelled_as` A string explaining the way to handle labelled vectors. Currently `"factor"` (the default) is the only possibility. This means that labelled variables that appear factor-like (by having a non-`NULL` `labels`-attribute) will be treated as factors, while other labelled variables will be treated as whatever base variable class they inherit from.

`mode` Vector of tasks to perform among the three categories "summarize", "visualize" and "check". The default, `c("summarize","visualize","check")`, implies that all three steps are performed. The steps selected in `mode` will be performed for each variable in `data` and their results are presented in the second part of the outputtet data cleaning overview document. The "summarize" step is responsible for creating the summary table, the "visualize" step is responsible for creating the plot and the "check" step is responsible for performing checks on the variable and printing the results if any problems are found.

`smartNum` If `TRUE` (the default), numeric and integer variables with less than 5 unique values are treated as factor variables in the checking, visualization and summary steps, and a message notifying the reader of this is printed in the data summary.

`preChecks` Vector of function names for check functions used in the pre-check stage. The pre-check stage consists of variable checks that should be performed before the summary/visualization/checking step. If any of these checks find problems, the variable will not be summarized nor visualized nor checked.

`file` The filename of the outputted rmarkdown (.Rmd) file. If set to `NULL` (the default), the filename will be the name of `data` prefixed with "dataMaid_", if this qualifies as a valid file name (e.g. no special characters allowed). Otherwise, `clean()` tries to create a valid filename by substituing illegal characters. Note that a valid file is of type .Rmd, hence all filenames should have a ".Rmd"-suffix.

`replace` If `FALSE` (the default), an error is thrown if one of the files that we are about to be created (.Rmd overview file and possible also a .html or .pdf file) already exist. If `TRUE`, no checks are performed and files on disc thus might be overwritten.

`vol` Extra text string or numeric that is appended on the end of the output file name(s). For example, if the dataset is called "myData", no file argument is supplied and `vol=2`, the output file will be called "dataMaid_myData2.Rmd"

`standAlone` A logical. If `TRUE`, the document begins with a markdown YAML preamble such that it can be rendered as a stand alone rmarkdown file, e.g. by calling `render`. If `FALSE`, this preamble is removed. Moreover, no matter the input to the `render` argument, the document will now not be rendered, as it has no preamble.

`twoCol` A logical. Should the results from the *summarize* and *visualize* steps be presented in two columns? Defaults to `TRUE`.

`quiet` A logical. If `TRUE` (the default), only a few messages are printed to the screen as `clean` runs. If `FALSE`, no messages are suppressed. The third option,

silent, renders the function completely silent, such that only fatal errors are printed.

openResult A logical. If TRUE (the default), the last file produced by clean is automatically opened by the end of the function run. This means that if render = TRUE, the rendered pdf or html file is opened, while if render = FALSE, the .Rmd file is opened.

characterChecks
A vector of the names of error-checking functions to apply to character vectors.

factorChecks A vector of the names of error-checking functions to apply to integer vectors.

labelledChecks
A vector of the names of error-checking functions to apply to character vectors.

numericChecks
A vector of the names of error-checking functions to apply to numeric vectors.

integerChecks
A vector of the names of error-checking functions to apply to integer vectors.

logicalChecks
A vector of the names of error-checking functions to apply to logical vectors.

dateChecks A vector of the names of error-checking functions to apply to Date vectors.

allChecks Vector of function names that should be used as check-functions for all variable types. Note that this argument overwrites the arguments characterChekcs, factorChecks, etc.

characterSummaries
A vector of the names of summary functions to apply to character vectors.

factorSummaries
A vector of the names of summary functions to apply to factor vectors.

labelledSummaries
A vector of the names of summary functions to apply to labelled vectors.

numericSummaries
A vector of the names of summary functions to apply to numeric vectors.

integerSummaries
A vector of the names of summary functions to apply to integer vectors.

logicalSummaries
A vector of the names of summary functions to apply to logical vectors.

dateSummaries
A vector of the names of summary functions to apply to Date vectors.

allSummaries Vector of function names that should be used as summary functions for all variable types. Note that this argument overwrites the arguments characterSummaries, factorSummaries, etc.

allVisuals A single function name. This funtion name is called for creating the plots for each variable in the "visualize" step. The default, "standardVisual" thus calls the visualFunction standardVisual for each variable in data.

listChecks A logical. Controls whether what checks that were used for each possible variable type are summarized in the output. Defaults to TRUE.

`maxProbVals` A positive integer or `Inf`. Maximum number of unique values printed from check-functions. In the case of `Inf`, all problematic values are printed. Defaults to `10`.

`maxDecimals` A positive integer or `Inf`. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If `Inf`, no rounding is performed.

`addSummaryTable`
A logical. If `TRUE` (the default), a summary table of the variable checks is added between the Data Cleaning Summary and the Variable List.

`reportTitle` A text string. If supplied, this will be the printed title of the report. If left unspecified, the title with the name of the supplied dataset.

`treatXasY` A list that indicates how non-standard variable classes should be treated. This parameter allows you to include variables that are not of class `factor`, `character`, `labelled`, `numeric`, `integer`, `logical` nor `Date` (or a class that inherits from any of these classes). The names of the list are the new classes and the entries are the names of the class, they should be treated as. If `clean()` should e.g. treat variables of class `raw` as characters and variables of class `complex` as numeric, you should put `treatXasY = list(raw = "character", complex = "numeric")`.

`...` FIX ME——— Other arguments that are passed on the to precheck, checking, summary and visualization functions.WHAT ARGUMENTS ARE RELEVANT TO MENTION HERE? ———- FIX ME

Details

Run a set of class-specific validation checks to check the variables in a dataset for potential errors. Performs checking steps according to user input and/or data type of the inputted variable. The checks are saved to an R markdown file which can rendered into an easy-to-read document. This document also includes summaries and visualizations of each variable in the dataset.

For each variable, a set of pre-check (controlled by the `preChecks` argument) is first run and then then a battery of functions are applied depending on the variable class. For each variable type the summarize/visualize/check functions are applied and and the results are written to an R markdown file.

Value

The function does not return anything. Its side effect (the production of a data cleaning overview document) is the reason for running the function.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (`https://doi.org/10.18637/jss.v090.i06`).

Examples

```
data(testData)
data(toyData)

check(toyData)

 ## Not run:
DF <- data.frame(x = 1:15)
clean(DF)

## End(Not run)

## Not run:
data(testData)
clean(testData)

## End(Not run)

# Overwrite any existing files generated by clean
## Not run:
clean(testData, replace=TRUE)

## End(Not run)

# Only include problematic variables in the output document
## Not run:
clean(testData, replace=TRUE, onlyProblematic=TRUE)

## End(Not run)

# Add user defined check-function to the checks performed on character variables:
# Here we add functionality to search for the string wally (ignoring case)
## Not run:
wheresWally <- function(v, ...) {
     res <- grepl("wally", v, ignore.case=TRUE)
     problem <- any(res)
     message <- "Wally was found in these data"
     checkResult(list(problem = problem,
                      message = message,
                      problemValues = v[res]))
}

wheresWally <- checkFunction(wheresWally,
                             description = "Search for the string 'wally' ignoring case",
                             classes = c("character")
                             )
# Add the newly defined function to the list of checks used for characters.
clean(testData, characterChecks=c(defaultCharacterChecks(), "wheresWally"),
      replace=TRUE)

## End(Not run)
```

```
#Handle non-supported variable classes using treatXasY: treat raw as character and
#treat complex as numeric. We also add a list variable, but as lists are not
#handled through treatXasY, this variable will be caught in the preChecks and skipped:
## Not run:
toyData$rawVar <- as.raw(c(1:14, 1))
toyData$compVar <- c(1:14, 1) + 2i
toyData$listVar <- as.list(c(1:14, 1))
clean(toyData, replace  = TRUE, treatXasY = list(raw = "character", complex = "numeric"))

## End(Not run)
```

`countMissing` *Summary function for missing values*

Description

A `summaryFunction`, intended to be called from `summarize` (and `makeDataReport`), which counts the number of missing (`NA`) values in a variable.

Usage

```
countMissing(v, ...)
```

Arguments

`v`	A variable (vector).
`...`	Not in use.

Value

A `summaryResult` object with the following entries: `$feature` ("No. missing obs."), `$result` (the number and percentage missing observations) and `$value` (the number of missing observations).

See Also

`summarize`, `allSummaryFunctions`, `summaryFunction`, `summaryResult`

Examples

```
countMissing(c(1:100, rep(NA, 10)))
```

defaultCharacterChecks

Default checks for character variables

Description

Default options for which checks to perform on character type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultCharacterChecks(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

defaultCharacterSummaries

Default summary functions for character variables

Description

Default options for which summaries to apply on character type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultCharacterSummaries(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

variableType, countMissing, uniqueValues, centralValue

Examples

```
#remove "variableType" from the summaries:
defaultCharacterSummaries(remove = "variableType")
```

---

defaultDateChecks *Default checks for Date variables*

Description

Default options for which checks to perform on Date type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultDateChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

---

defaultDateSummaries *Default summary functions for Date variables*

Description

Default options for which summaries to apply on Date type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultDateSummaries(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

variableType, countMissing, uniqueValues, centralValue, minMax, quartiles

Examples

```
defaultDateSummaries()
```

---

defaultFactorChecks

Default checks for factor variables

Description

Default options for which checks to perform on factor type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultFactorChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

defaultFactorSummaries

Default summary functions for factor variables

Description

Default options for which summaries to apply on factor type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultFactorSummaries(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

codevariableType, `countMissing`, `uniqueValues`, `centralValue`

Examples

```
#remove "countMissing" for the summaries:
defaultFactorSummaries(remove = "countMissing")
```

defaultHavenlabelledChecks

Default checks for haven_labelled variables

Description

Default options for which checks to perform on haven_labelled type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultHavenlabelledChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

---

defaultHavenlabelledSummaries

Default summary functions for haven_labelled variables

Description

Default options for which summaries to apply on haven_labelled type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultHavenlabelledSummaries(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

`variableType`, `countMissing`, `uniqueValues`, `centralValue`

Examples

```
#remove "centralValue":
defaultHavenlabelledSummaries(remove = "centralValue")
```

`defaultIntegerChecks`
Default checks for integer variables

Description

Default options for which checks to perform on integer type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultIntegerChecks(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

`defaultIntegerSummaries`
Default summary functions for integer variables

Description

Default options for which summaries to apply on integer type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultIntegerSummaries(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

variableType, countMissing, uniqueValues, centralValue, quartiles, minMax

Examples

```
#remove "countMissing":
defaultIntegerSummaries(remove = "countMissing")
```

---

defaultLabelledChecks

Default checks for labelled variables

Description

Default options for which checks to perform on labelled type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultLabelledChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

---

defaultLabelledSummaries

Default summary functions for labelled variables

Description

Default options for which summaries to apply on labelled type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultLabelledSummaries(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

variableType, countMissing, uniqueValues, centralValue

Examples

```
#remove "centralValue":
defaultLabelledSummaries(remove = "centralValue")
```

---

defaultLogicalChecks

Default checks for logical variables

Description

Default options for which checks to perform on logical type variables in check and makeDataReport, possibly user-modified by adding extra function names using add or removing default function names with remove.

Usage

```
defaultLogicalChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

defaultLogicalSummaries

Default summary functions for logical variables

Description

Default options for which summaries to apply on logical type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultLogicalSummaries(remove = NULL, add = NULL)
```

Arguments

`remove`	Character vector of function names. Checks to remove from the returned vector
`add`	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

`variableType`, `countMissing`, `uniqueValues`, `centralValue`

Examples

```
#remove "uniqueValues":
defaultLogicalSummaries(remove = "uniqueValues")
```

defaultNumericChecks

Default checks for numeric variables

Description

Default options for which checks to perform on numeric type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultNumericChecks(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A vector of function names.

---

defaultNumericSummaries

Default summary functions for numeric variables

Description

Default options for which summaries to apply on numeric type variables in `check` and `makeDataReport`, possibly user-modified by adding extra function names using `add` or removing default function names with `remove`.

Usage

```
defaultNumericSummaries(remove = NULL, add = NULL)
```

Arguments

remove	Character vector of function names. Checks to remove from the returned vector
add	Character vector of function names. Checks to add to the returned vector

Value

A list of function names (as character strings).

See Also

`variableType`, `countMissing`, `uniqueValues`, `centralValue`, `quartiles`, `minMax`

Examples

```
#remove "uniqueValues":
defaultNumericSummaries(remove = "uniqueValues")
```

description *Extract the contents of the attribute* description

Description

If the object, x, is itself of class checkFunction, summaryFunction or visualFunction, the contents of x's attribute description is returned. Otherwise, NULL is returned.

Usage

```
description(x)

description(x) <- value
```

Arguments

x	The object for which the description attribute should be extracted.
value	New value

Value

A description of what x does, given as a character string.

Examples

```
#Extract the description of the checkFunction identifyMissing
description(identifyMissing)

#Extract the description of the summaryFunction minMax
description(minMax)

#Extract the description of the visualFunction basicVisual
description(basicVisual)
```

exampleData *Example data with zero-inflated variables*

Description

An artificial dataset, intended for presenting the extended features of dataMaid, which is a toolset for identifying potential errors in a dataset.

Usage

```
exampleData
```

Format

A `data.frame` with 300 observations on the following 6 variables.

`addresses` a factor with fictitious US addresses

`binomial` a numeric vector with a binomial distributed variable

`poisson` a numeric vector with a Poisson distributed variable

`gauss` a numeric vector with a Gaussian distributed variable

`zigauss` a numeric vector with a zero-inflated Gaussian distributed variable

`bpinteraction` a factor with interactions between binomial and poisson values

Source

Artificial data

Examples

```
## Not run:
isID <- function(v, nMax = NULL, ...) {
  out <- list(problem = FALSE, message = "")
  if (class(v) %in% c("character", "factor", "labelled", "numeric", "integer")) {
    v <- as.character(v)
    lengths <- nchar(v)
    if (all(lengths > 10) & length(unique(lengths)) == 1) {
      out$problem <- TRUE
      out$message <- "Warning: This variable seems to contain ID codes!"
    }
  }
  out
}


countZeros <- function(v, ...) {
  res <- length(which(v == 0))
  summaryResult(list(feature = "No. zeros", result = res, value = res))
}
countZeros <- summaryFunction(countZeros, description = "Count number of zeros",
                              classes = allClasses())
summarize(toyData, numericSummaries = c(defaultNumericSummaries()))


mosaicVisual <- function(v, vnam, doEval) {
  thisCall <- call("mosaicplot", table(v), main = vnam, xlab = "")
  if (doEval) {
    return(eval(thisCall))
  } else return(deparse(thisCall))
}
mosaicVisual <- visualFunction(mosaicVisual,
                               description = "Mosaic plots using graphics",
                               classes = allClasses())
```

```
identifyColons <- function(v, nMax = Inf, ... ) {
  v <- unique(na.omit(v))
  problemMessage <- "Note: The following values include colons:"
  problem <- FALSE
  problemValues <- NULL

  problemValues <- v[sapply(gregexpr("[[:xdigit:]]:[[:xdigit:]]", v),
                            function(x) all(x != -1))]

  if (length(problemValues) > 0) {
    problem <- TRUE
  }

  problemStatus <- list(problem = problem,
                        problemValues = problemValues)
  outMessage <- messageGenerator(problemStatus, problemMessage, nMax)

  checkResult(list(problem = problem,
                   message = outMessage,
                   problemValues = problemValues))
}

identifyColons <- checkFunction(identifyColons,
                                description = "Identify non-suffixed nor -prefixed colons",
                                classes = c("character", "factor", "labelled"))
makeDataReport(exampleData, replace = TRUE,
      preChecks = c("isKey", "isEmpty", "isID"),
      allVisuals = "mosaicVisual",
      characterSummaries = c(defaultCharacterSummaries(), "countZeros"),
      factorSummaries = c(defaultFactorSummaries(), "countZeros"),
      labelledSummaries = c(defaultLabelledSummaries(), "countZeros"),
      numericSummaries = c(defaultNumericSummaries(), "countZeros"),
      integerSummaries = c(defaultIntegerSummaries(), "countZeros"),
      characterChecks = c(defaultCharacterChecks(), "identifyColons"),
      factorChecks = c(defaultFactorChecks(), "identifyColons"),
      labelledCheck = c(defaultLabelledChecks(), "identifyColons"))




## End(Not run)
```

`identifyCaseIssues` *A checkFunction for identifying case issues*

Description

A `checkFunction` to be called from `check` that identifies values in a vector that appear multiple times with different case settings.

Usage

```
identifyCaseIssues(v, nMax = 10)
```

Arguments

v	A character, factor, haven_labelled or labelled variable to check.
nMax	The maximum number of problematic values to report. Default is 10. Set to Inf if all problematic values are to be included in the outputted message, or to 0 for no output.

Value

A checkResult with three entires: $problem (a logical indicating whether case issues where found), $message (a message describing which values in v resulted in case issues) and $problemValues (the problematic values in their original format). Note that Only unique problematic values are listed and they are presented in alphabetical order.

See Also

check, allCheckFunctions, checkFunction, checkResult

Examples

```
identifyCaseIssues(c("val", "b", "1", "1", "vAl", "VAL", "oh", "OH"))
```

---

identifyLoners *A checkFunction for identifying sparsely represented values (loners)*

Description

A checkFunction to be called from check that identifies values that only occur less than 6 times in factor, (haven_)labelled, or character variables (that is, loners).

Usage

```
identifyLoners(v, nMax = 10)
```

Arguments

v	A character, (haven_)labelled, or factor variable to check.
nMax	The maximum number of problematic values to report. Default is 10. Set to Inf if all problematic values are to be included in the outputted message, or to 0 for no output.

Details

For character, (haven_)labelled, and factor variables, identify values that only have a very low number of observations, as these categories might be problematic when conducting an analysis. Unused factor levels are not considered "loners". "Loners" are defined as values with 5 or less observations, reflecting the commonly use rule of thumb for performing chi squared tests.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether case issues where found), `$message` (a message describing which values in `v` were loners) and `$problemValues` (the problematic values in their original format). Note that Only unique problematic values are listed and they are presented in alphabetical order.

See Also

`check`, `allCheckFunctions`, `checkFunction`, `checkResult`

Examples

```
identifyLoners(c(rep(c("a", "b", "c"), 10), "d", "d"))
```

---

identifyMissing *A checkFunction for identifying miscoded missing values.*

Description

A checkFunction to be called from `check` that identifies values that appear to be miscoded missing values.

Usage

```
identifyMissing(v, nMax = 10, ...)
```

Arguments

`v`	A variable to check.
`nMax`	The maximum number of problematic values to report. Default is `10`. Set to `Inf` if all problematic values are to be included in the outputted message, or to `0` for no output.
`...`	Not in use.

Details

identifyMissing tries to identify common choices of missing values outside of the R standard (NA). These include special words (NaN and Inf (no matter the cases)), one or more -9/9's (e.g. 999, "99", -9, "-99"), one ore more -8/8's (e.g. -8, 888, -8888), Stata style missing values (commencing with ".") and other character strings ("", " ", "-", "NA" miscoded as character). If the variable is numeric/integer or a character/factor variable consisting only of numbers and with more than 11 different values, the numeric miscoded missing values (999, 888, -99, -8 etc.) are only recognized as miscoded missing if they are maximum or minimum, respectively, and the distance between the second largest/smallest value and this maximum/minimum value is greater than one.

Value

A checkResult with three entires: $problem (a logical indicating whether midcoded missing values where found), $message (a message describing which values in v were suspected to be miscoded missing values), and $problemValues (the problematic values in their original format). Note that Only unique problematic values are listed and that they are presented in alphabetical order.

See Also

check, allCheckFunctions, checkFunction, checkResult

Examples

```
##data(testData)
##testData$miscodedMissingVar
##identifyMissing(testData$miscodedMissingVar)

#Identify miscoded numeric missing values
v1 <- c(1:15, 99)
v2 <- c(v1, 98)
v3 <- c(-999, v2, 9999)
identifyMissing(v1)
identifyMissing(v2)
identifyMissing(v3)
identifyMissing(factor(v3))
```

---

identifyNums *A checkFunction*

Description

A checkFunction to be called from check for identifying numeric variables that have been misclassified as categorical.

Usage

```
identifyNums(v, nVals = 12, ...)
```

Arguments

v	A character, factor, or (haven_)labelled variable to check.
nVals	An integer determining how many unique values a variable must have before it can potentially be determined to be a misclassified numeric variable. The default is 12.
...	Not in use.

Details

A categorical variable is suspected to be a misclassified numeric variable if it has the following two properties: First, it should consist exclusively of numbers (possibly including signs and decimals points). Secondly, it must have at least nVals unique values. The default values of nVals is 12, which means that e.g. variables including answers on a scale from 0-10 will not be recognized as misclassified numerics.

Value

A checkResult with three entires: $problem (a logical indicating the variable is suspected to be a misclassified numeric variable), $message (if a problem was found, the following message: "Note: The variable consists exclusively of numbers and takes a lot of different values. Is it perhaps a misclassified numeric variable?", otherwise "") and $problemValues (always NULL).

See Also

check, allCheckFunctions, checkFunction, checkResult

Examples

```
#Positive and negative numbers, saved as characters
identifyNums(c(as.character(-9:9)))

#An ordinary character variable
identifyNums(c("a", "b", "c", "d", "e.f", "-a", 1:100))
```

---

identifyOutliers *A checkFunction for identifying outliers*

Description

A checkFunction to be called from check that identifies outlier values in a Date/numeric/integer variable.

Usage

```
identifyOutliers(v, nMax = 10, maxDecimals = 2)
```

Arguments

v	A Date, numeric or integer variable to check.
nMax	The maximum number of problematic values to report. Default is 10. Set to Inf if all problematic values are to be included in the outputted message, or to 0 for no output.
maxDecimals	A positive integer or Inf. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If Inf, no rounding is performed.

Details

Outliers are identified based on an outlier rule that is appropriate for asymmetric data. Outliers are observations outside the range

$$Q1 - 1.5 * exp(a * MC) * IQR; Q3 + 1.5 * exp(b * MC) * IQR$$

where Q1, Q3, and IQR are the first quartile, third quartile, and inter-quartile range, MC is the 'medcouple', a robust concept and estimator of skewness, and a and b are appropriate constants (-4 and 3). The medcouple is defined as a scaled median difference of the left and right half of distribution, and hence not based on the third moment as the classical skewness.

When the data are symmetric, the measure reduces to the standard outlier rule also used in Tukey Boxplots (consistent with the boxplot function), i.e. as values that are smaller than the 1st quartile minus the inter quartile range (IQR) or greater than the third quartile plus the IQR.

For Date variables, the calculations are done on their raw numeric format (as obtained by using unclass), after which they are translated back to Dates. Note that no rounding is performed for Dates, no matter the value of maxDecimals.

Value

A checkResult with three entires: $problem (a logical indicating whether outliers were found), $message (a message describing which values are outliers) and $problemValues (the outlier values).

See Also

check, allCheckFunctions, checkFunction, checkResult, mc

Examples

```
identifyOutliers(c(1:10, 200, 200, 700))
```

identifyOutliersTBStyle *A checkFunction for identifying outliers Turkey Boxstole style*

Description

A checkFunction to be called from `check` that identifies outlier values in a numeric/integer/Date variable by use of the Turkey Boxplot method (consistent witht the `boxplot` function).

Usage

```
identifyOutliersTBStyle(v, nMax = 10, maxDecimals = 2)
```

Arguments

`v`	A numeric, integer or Date variable to check.
`nMax`	The maximum number of problematic values to report. Default is `10`. Set to `Inf` if all problematic values are to be included in the outputted message, or to `0` for no output.
`maxDecimals`	A positive integer or `Inf`. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If `Inf`, no rounding is performed.

Details

Outliers are defined in the style of Turkey Boxplots (consistent with the `boxplot` function), i.e. as values that are smaller than the 1st quartile minus the inter quartile range (IQR) or greater than the third quartile plus the IQR.

For Date variables, the calculations are done on their raw numeric format (as obtained by using `unclass`), after which they are translated back to Dates. Note that no rounding is performed for Dates, no matter the value of `maxDecimals`.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether outliers were found), `$message` (a message describing which values are outliers) and `$problemValues` (the outlier values).

See Also

`check`, `allCheckFunctions`, `checkFunction`, `checkResult`

Examples

```
identifyOutliersTBStyle(c(1:10, 200, 200, 700))
```

identifyWhitespace *A checkFunction for identifying whitespace*

Description

A checkFunction to be called from `check` that identifies prefixed and suffixed whitespace(s) in character, (haven_)labelled or factor variables.

Usage

```
identifyWhitespace(v, nMax = 10)
```

Arguments

`v`	A character, (haven_)labelled or factor variable to check.
`nMax`	The maximum number of problematic values to report. Default is `10`. Set to `Inf` if all problematic values are to be included in the outputted message, or to `0` for no output.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether any whitespaces were fount), `$message` (a message describing which values were prefixed or suffixed with whitespace) and `$problemValues` (the problematic values). Note that only unique values are printed in the message, and that they are sorted alphabetically.

See Also

`check`, `allCheckFunctions`, `checkFunction`, `checkResult`

Examples

```
identifyWhitespace(c("a", " b", "c", "d ", "e  "))
```

isCPR *Check if a variable consists of Danish CPR numbers*

Description

A `checkFunction` that checks if `v` consists exclusively of valid Danish civil registration (CPR) numbers, ignoring missing values. This function is intended for use as a precheck in `makeDataReport`, ensuring that CPR numbers are not included in a `dataMaid` output document.

Usage

```
isCPR(v, ...)
```

Arguments

v	A variable (vector) to check. This variable is allowed to have any class.
...	Not in use.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether the variable consists of CPR numbers), `$message` (if a problem was found, the following message: "Warning: The variable seems to consist of Danish civil registration (CPR) numbers.", otherwise "") and `$problemValues` (always `NULL`).

See Also

`check`, `allCheckFunctions`, `checkFunction`, `checkResult`

Examples

```
CPRs <-  sapply(c("01011988", "02011987", "04052006", "01021990", "01021991",
                  "01021993", "01021994", "01021995", "01021996", "01021997",
                  "01021970", "01021971", "01021972", "01021973", "01021974"), dataMaid:::ma
nonCPRs <- c(1:10)
mixedCPRs <- c(CPRs, nonCPRs)

#identify problem
isCPR(CPRs)

#no problem as there are no CPRs
isCPR(nonCPRs)

#no problem because not ALL values are CPRs
isCPR(mixedCPRs)
```

---

isKey *Check if a variable qualifies as a key*

Description

A `checkFunction` that checks if `v` is a key, that is, if every observation has a unique value in `v` and `v` is not a numeric/integer nor a Date variable. This function is intended for use as a precheck in `makeDataReport`.

Usage

```
isKey(v)
```

Arguments

v	A variable (vector) to check. All variable types are allowed.

Details

Note that numeric or integer variables are not considered candidates for keys, as truly continuous measurements will most likely result in unique values for each observation.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether `v` is a key), `$message` (if a problem was found, the following message: "The variable is a key (distinct values for each observation).", otherwise "") and `$problemValues` (always `NULL`).

See Also

`check`, `allCheckFunctions`, `checkFunction`, `checkResult`

Examples

```
keyVar <- c("a", "b", "c", "d", "e", "f")
notKeyVar <- c("a", "a", "b", "c", "d", "e", "f")

isKey(keyVar)
isKey(notKeyVar)
```

`isSingular` *Check if a variable only contains a single value*

Description

A `checkFunction` that checks if `v` only contains a single unique value, aside from missing values. This function is intended for use as a precheck in `makeDataReport`.

Usage

```
isSingular(v)

isEmpty(v)
```

Arguments

`v`	A variable (vector) to check. All variable types are allowed.

Value

A `checkResult` with three entires: `$problem` (a logical indicating whether `v` contains only one value), `$message` (if a problem was found, a message describing which single value the variable takes and how many missing observations it contains, otherwise ""), and `$problemValues` (always `NULL`).

See Also

check, allCheckFunctions, checkFunction, checkResult

Examples

```
singularVar <- c(rep("a", 10), NA, NA)
notSingularVar <- c("a", "a", "b", "c", "d", "e", "f", NA, NA)

isSingular(singularVar)
isSingular(notSingularVar)
```

---

isSupported *Check if a variable has a class supported by dataMaid*

Description

A checkFunction that checks if v has one of the classes supported by dataMaid, namely character, factor, numeric, integer, labelled, haven_labelled, logical and Date (inlcuding other classes that inherits from any of these classes). A user supported list can be provided in the treatXasY argument, which will let the user decide how unsupported classes should be treated. This function is intended for use as a precheck in makeDataReport.

Usage

```
isSupported(v)
```

Arguments

v	A variable (vector) to check. All variable types are allowed.

Value

A checkResult with three entires: $problem (a logical indicating whether v contains only one value), $message (if a problem was found, a message describing which single value the variable takes and how many missing observations it contains, otherwise ""), and $problemValues (always NULL).

See Also

check, allCheckFunctions, checkFunction, checkResult

Examples

```
integerVar <- 1:10 #supported
rawVar <- as.raw(1:10) #not supported

isSupported(integerVar)
isSupported(rawVar)
```

makeCodebook *Produce a data codebook*

Description

Make a data codebook that summarizes the contents of a dataset. The result is saved to an R markdown file which can rendered into an easy-to-read codebook in pdf, html or word formats.

Usage

```
makeCodebook(data, vol = "", reportTitle = NULL, file = NULL, ...)
```

Arguments

data	The dataset to be checked. This dataset should be of class data.frame, tibble or matrix. If it is of classs matrix, it will be converted to a data.frame.
vol	Extra text string or numeric that is appended on the end of the output file name(s). For example, if the dataset is called "myData", no file argument is supplied and vol=2, the output file will be called "codebook_myData2.Rmd"
reportTitle	A text string. If supplied, this will be the printed title of the report. If left unspecified, the title with the name of the supplied dataset.
file	The filename of the outputted rmarkdown (.Rmd) file. If set to NULL (the default), the filename will be the name of data prefixed with "codebook_", if this qualifies as a valid file name (e.g. no special characters allowed). Otherwise, makeCodebook() tries to create a valid filename by substituing illegal characters. Note that a valid file is of type .Rmd, hence all filenames should have a ".Rmd"-suffix.
...	Additional parameters passed to makeDataReport.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (https://doi.org/10.18637/jss.v090.i06).

makeDataReport *Produce a data report*

Description

Make a data overview report that summarizes the contents of a dataset and flags potential problems. The potential problems are identified by running a set of class-specific validation checks, so that different checks are performed on different variables types. The checking steps can be customized according to user input and/or data type of the inputted variable. The checks are saved to an R markdown file which can rendered into an easy-to-read data report in pdf, html or word formats. This report also includes summaries and visualizations of each variable in the dataset.

Usage

```
makeDataReport(data, output = NULL, render = TRUE, useVar = NULL,
  ordering = c("asIs", "alphabetical"), onlyProblematic = FALSE,
  labelled_as = c("factor"), mode = c("summarize", "visualize",
  "check"), smartNum = TRUE, preChecks = c("isKey", "isSingular",
  "isSupported"), file = NULL, replace = FALSE, vol = "",
  standAlone = TRUE, twoCol = TRUE, quiet = TRUE,
  openResult = TRUE, summaries = setSummaries(),
  visuals = setVisuals(), checks = setChecks(), listChecks = TRUE,
  maxProbVals = 10, maxDecimals = 2, addSummaryTable = TRUE,
  codebook = FALSE, reportTitle = NULL, treatXasY = NULL,
  includeVariableList = TRUE, ...)
```

Arguments

`data`	The dataset to be checked. This dataset should be of class `data.frame`, `tibble` or `matrix`. If it is of classs `matrix`, it will be converted to a `data.frame`.
`output`	Output format. Options are `"pdf"`, `"word"` (.docx) and `"html"`. If `NULL` (the default), the output format depends two sequential checks. First, whether a LaTeX installation is available, in which case `pdf` output is chosen. Secondly, if no LaTeX installation is found, then if the operating system is Windows, `word` output is used. Lastly, if neither of these checks are positive, `html` output is used.
`render`	Should the output file be rendered (defaults to `TRUE`), i.e. should a pdf/word/html document be generated and saved to the disc?
`useVar`	Variables to describe in the report. If `NULL` (the default), all variables in `data` are included. If a vector of variable names is supplied, only the variables in `data` that are also in `useVar` are included in the data report.
`ordering`	Choose the ordering of the variables in the variable presentation. The options are "asIs" (ordering as in the dataset) and "alphabetical" (alphabetical order).
`onlyProblematic`	A logical. If `TRUE`, only the variables flagged as problematic in the check step will be included in the variable list.
`labelled_as`	A string explaining the way to handle labelled and haven_labelled vectors. Currently `"factor"` (the default) is the only possibility. This means that labelled or haven_labelled variables that appear factor-like (by having a non-`NULL` `labels`-attribute) will be treated as factors, while other labelled or haven_labelled variables will be treated as whatever base variable class they inherit from.
`mode`	Vector of tasks to perform among the three categories "summarize", "visualize" and "check". The default, `c("summarize","visualize","check")`, implies that all three steps are performed. The steps selected in `mode` will be performed for each variable in `data` and their results are presented in the second part of the outputtet data report. The "summarize" step is responsible for creating the summary table, the "visualize" step is responsible for creating the plot and the "check" step is responsible for performing checks on the variable and printing the results if any problems are found.

smartNum — If TRUE (the default), numeric and integer variables with less than 5 unique values are treated as factor variables in the checking, visualization and summary steps, and a message notifying the reader of this is printed in the data summary.

preChecks — Vector of function names for check functions used in the pre-check stage. The pre-check stage consists of variable checks that should be performed before the summary/visualization/checking step. If any of these checks find problems, the variable will not be summarized nor visualized nor checked.

file — The filename of the outputted rmarkdown (.Rmd) file. If set to NULL (the default), the filename will be the name of data prefixed with "dataMaid_", if this qualifies as a valid file name (e.g. no special characters allowed). Otherwise, makeDataReport() tries to create a valid filename by substituing illegal characters. Note that a valid file is of type .Rmd, hence all filenames should have a ".Rmd"-suffix.

replace — If FALSE (the default), an error is thrown if one of the files that we are about to be created (.Rmd overview file and possible also a .html, .pdf or .docx file) already exist. If TRUE, no checks are performed and files on disc thus might be overwritten.

vol — Extra text string or numeric that is appended on the end of the output file name(s). For example, if the dataset is called "myData", no file argument is supplied and vol=2, the output file will be called "dataMaid_myData2.Rmd"

standAlone — A logical. If TRUE, the document begins with a markdown YAML preamble such that it can be rendered as a stand alone rmarkdown file, e.g. by calling render. If FALSE, this preamble is removed. Moreover, no matter the input to the render argument, the document will now not be rendered, as it has no preamble.

twoCol — A logical. Should the results from the *summarize* and *visualize* steps be presented in two columns? Defaults to TRUE.

quiet — A logical. If TRUE (the default), only a few messages are printed to the screen as makeDataReport runs. If FALSE, no messages are suppressed. The third option, silent, renders the function completely silent, such that only fatal errors are printed.

openResult — A logical. If TRUE (the default), the last file produced by makeDataReport is automatically opened by the end of the function run. This means that if render = TRUE, the rendered pdf, word or html file is opened, while if render = FALSE, the .Rmd file is opened.

summaries — A list of summaries to use on each supported variable type. We recommend using setSummaries for creating this list and refer to the documentation of this function for more details.

visuals — A list of visual functions to use on each supported variable type. We recommend using setVisuals for creating this list and refer to the documentation of this function for more details.

checks — A list of checks to use on each supported variable type. We recommend using setChecks for creating this list and refer to the documentation of this function for more details.

listChecks — A logical. Controls whether what checks that were used for each possible variable type are summarized in the output. Defaults to TRUE.

maxProbVals A positive integer or Inf. Maximum number of unique values printed from check-functions. In the case of Inf, all problematic values are printed. Defaults to 10.

maxDecimals A positive integer or Inf. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If Inf, no rounding is performed.

addSummaryTable A logical. If TRUE (the default), a summary table of the variable checks is added between the Data Cleaning Summary and the Variable List. Only one of addSummaryTable and addCodebookTable can be TRUE.

codebook A logical. Defaults to FALSE. If TRUE then the document is tweaked to better represent a codebook.

reportTitle A text string. If supplied, this will be the printed title of the report. If left unspecified, the title with the name of the supplied dataset.

treatXasY A list that indicates how non-standard variable classes should be treated. This parameter allows you to include variables that are not of class factor, character, labelled, haven_labelled, numeric, integer, logical nor Date (or a class that inherits from any of these classes). The names of the list are the new classes and the entries are the names of the class, they should be treated as. If makeDataReport() should e.g. treat variables of class raw as characters and variables of class complex as numeric, you should put treatXasY = list(raw = "character", complex = "numeric").

includeVariableList A logical indicating whether the results of the summarize/visualize/check-steps should be added to the report. Defaults to TRUE. Note that setting it to FALSE does currently not speed up computations, it just means that the information is not printed in the report.

... Other arguments that are passed on the to precheck, checking, summary and visualization functions.

Details

For each variable, a set of pre-check functions (controlled by the preChecks argument) are first run and then then a battery of functions are applied depending on the variable class. For each variable type the summarize/visualize/check functions are applied and and the results are written to an R markdown file.

Value

The function does not return anything. Its side effect (the production of a data report) is the reason for running the function.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (https://doi.org/10.18637/jss.v090.i06).

Examples

```
data(testData)
data(toyData)

check(toyData)

 ## Not run:
DF <- data.frame(x = 1:15)
makeDataReport(DF)

## End(Not run)

## Not run:
data(testData)
makeDataReport(testData)

## End(Not run)

# Overwrite any existing files generated by makeDataReport
## Not run:
makeDataReport(testData, replace=TRUE)

## End(Not run)

# Change output format to Word/docx:
## Not run:
makeDataReport(testData, replace=TRUE, output = "word")

## End(Not run)

# Only include problematic variables in the output document
## Not run:
makeDataReport(testData, replace=TRUE, onlyProblematic=TRUE)

## End(Not run)

# Add user defined check-function to the checks performed on character variables:
# Here we add functionality to search for the string wally (ignoring case)
## Not run:
wheresWally <- function(v, ...) {
     res <- grepl("wally", v, ignore.case=TRUE)
     problem <- any(res)
     message <- "Wally was found in these data"
     checkResult(list(problem = problem,
                      message = message,
                      problemValues = v[res]))
}

wheresWally <- checkFunction(wheresWally,
                             description = "Search for the string 'wally' ignoring case",
                             classes = c("character")
                             )
```

```
# Add the newly defined function to the list of checks used for characters.
makeDataReport(testData,
      checks = setChecks(character = defaultCharacterChecks(with = "wheresWally")),
      replace=TRUE)

## End(Not run)

#Handle non-supported variable classes using treatXasY: treat raw as character and
#treat complex as numeric. We also add a list variable, but as lists are not
#handled through treatXasY, this variable will be caught in the preChecks and skipped:
## Not run:
toyData$rawVar <- as.raw(c(1:14, 1))
toyData$compVar <- c(1:14, 1) + 2i
toyData$listVar <- as.list(c(1:14, 1))
makeDataReport(toyData, replace  = TRUE,
    treatXasY = list(raw = "character", complex = "numeric"))

## End(Not run)
```

`messageGenerator` *Produce a message for the output of a checkFunction*

Description

Helper function for producing output messages for `checkFunction` type functions.

Usage

```
messageGenerator(problemStatus,
  message = "Note that a check function found the following problematic values:",
  nMax = 10)
```

Arguments

`problemStatus`	A list consisting of two entries: `$problem` - logical indicating whether a problem was found by the `checkFunction` responsible for the making the `messageGenerator()` call, `$problemValues` - a vector of values from the variable that were deemed problematic (see details below).
`message`	Optional, but recommended. A message describing what problem the problem values are related to. If `NULL` a standard message is added using the name of the function that called `messageGenerator`.
`nMax`	Maximum number of problem values to be printed in the message. If the total number of problem values exceeds nMax, the number of omitted problem values are added to the message. Defaults to `Inf`, in which case all problem values are printed.

Details

This function is a tool for building checkFunctions for the dataMaid makeDataReport function. checkFunctions will often identify a number of values in a variable that are somehow problematic. messageGenerator takes these values, pastes them together with a problem description and makes sure that the formatting is appropriate for being rendered in a rmarkdown document. We recommend writing short and precise problem descriptions (see examples), but if no message is supplied, the following message is generated: "Note that a check function found the following problematic values: [problem values]".

Value

A character string with a problem description.

See Also

check, checkFunction, makeDataReport

Examples

```
#Varibales with/without underscores
 noUSVar <- c(1:10)
 USVar <- c("_a", "n_b", "b_", "_", 1:10)

#Define a checkFunction using messageGenerator with a manual
#problem description:
identifyUnderscores <- function(v, nMax = Inf) {
  v <- as.character(v)
  underscorePlaces <- regexpr("_", v) > 0
  problemValues <- unique(v[underscorePlaces])
  problem <- any(underscorePlaces)
  message <- messageGenerator(list(problemValues = problemValues, problem = problem),
                              "The following values contain underscores:",
                              nMax = nMax)
  checkResult(list(problem = problem, message = message,
      problemValues = problemValues))
 }

 identifyUnderscores(noUSVar) #no problem
 identifyUnderscores(USVar) #problems

#Only print the first two problemvalues in the message:
 identifyUnderscores(USVar, nMax = 2)

#Define same function, but without a manual problem description in
#the messageGenerator-call:
 identifyUnderscores2 <- function(v, nMax = Inf) {
  v <- as.character(v)
  underscorePlaces <- regexpr("_", v) > 0
  problemValues <- unique(v[underscorePlaces])
  problem <- any(underscorePlaces)
```

```
 message <- messageGenerator(list(problemValues = problemValues,
                                  problem = problem), nMax = nMax)
 checkResult(list(problem = problem, message = message,
     problemValues = problemValues))
}

identifyUnderscores2(noUSVar) #no problem
identifyUnderscores2(USVar) #problems
```

`minMax` *summaryFunction for minimum and maximum*

Description

A `summaryFunction`, intended to be called from `summarize`, which returns the minimum and maximum values of a variable. NA, NaN and Inf values are removed prior to the computations.

Usage

```
minMax(v, maxDecimals = 2)
```

Arguments

`v`	A variable (vector) of type numeric or integer.
`maxDecimals`	A positive integer or `Inf`. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If `Inf`, no rounding is performed.

Value

An object of class `summaryResult` with the following entries: `$feature` ("Min. and max."), `$result` (the minimum and maximum of `v`), and `$value` (minimum and maximum in their orignial format).

See Also

`summaryFunction`, `summarize`, `summaryResult`, `allSummaryFunctions`

Examples

```
minMax(c(1:100))
```

`presidentData` *Semi-artificial data about the US presidents*

Description

A dataset with information about the first 45 US presidents as well as a 46th person, who is not a US president. The dataset was constructed to show the capabilities of `dataMaid` and therefore, it has been constructed to include errors and miscodings. Each observation in the dataset corresponds to a person. The dataset uses the non-standard class `Name` which is simply an attribute that has been added to two variables in order to show how `dataMaid` handles non-supported classes.

Usage

```
presidentData
```

Format

A data frame with 46 rows and 11 variables.

lastName A `Name` type variable containing the last name of the president.

firstName A `Name` type variable containing the first name of the president.

orderOfPresidency A factor variable indicating the order of the presidents (with George Washington as number 1 and Donald Trump as number 45).

birthday A Date variable with the birthday of the president

stateOfBirth A character variable with the state in which the president was born.

assassinationAttempt A numeric variable indicating whether there was an assassination attempt (`1`) or not (`0`) on the president.

sex A factor variable with the sex of the president.

ethnicity A factor variable with the ethnicity of the president.

presidencyYears A numeric variable with the duration of the presidency, in years.

ageAtInauguration A character variable with the age at inauguration.

favoriteNumber A `complex` type variable with a fictional favorite number for each president.

Source

Artificial dataset constructed based on the US president dataset available from Data Explorer[3].

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (`https://doi.org/10.18637/jss.v090.i06`).

[3] `http://www.data-explorer.com/data`

Examples

```
data(presidentData)
```

`quartiles` *summaryFunction for quartiles*

Description

A `summaryFunction`, intended to be called from `summarize`, which calculates the 1st and 3rd quartiles of a variable. NA, NaN and Inf values are removed prior to the computations.

Usage

```
quartiles(v, maxDecimals = 2)
```

Arguments

`v`	A variable (vector) of type numeric or integer.
`maxDecimals`	A positive integer or `Inf`. Number of decimals used when printing numerical values in the data summary and in problematic values from the data checks. If `Inf`, no rounding is performed.

Details

The quartiles are computed using the `quantile` function from `stats`, using type 7 quantiles for integer and numeric variables and type 1 quantiles for Date variables.

Value

An object of class `summaryResult` with the following entries: `$feature` ("1st and 3rd quartiles"), `$result` (the 1st and 3rd quartiles of `v`) and `$value` (the quartiles in their original format).

See Also

`summaryFunction`, `summarize`, `summaryResult`, `allSummaryFunctions`

Examples

```
quartiles(c(1:100))

quartiles(rnorm(1000), maxDecimals = 4)
```

refCat *summaryFunction that finds reference level for factor variables*

Description

A `summaryFunction`, intended to be called from `summarize`, which returns the reference level of a factor variable, i.e. the first category as returned by `levels(v)`. This level will serve as the reference category and get absorbed into the intercept for most standard model fitting procedures and therefore, it may be convenient to know.

Usage

```
refCat(v, ...)
```

Arguments

`v`	A variable (vector) of type factor.
`...`	Not in use.

Value

An object of class `summaryResult` with the following entries: `$feature` ("Reference level"), `$result` (the reference level of `v`), and `$value` (identical to result).

See Also

`summaryFunction`, `summarize`, `summaryResult`, `allSummaryFunctions`

Examples

```
refCat(factor(letters))
```

render *Simplified Rmarkdown rendering*

Description

Render a Rmarkdown (.Rmd) file, `file`, to the output format specified in its preamble. If no output format is specified, it will be rendered to html.

Usage

```
render(file, quiet)
```

Arguments

file	A character string path to the file that is to be rendered. This file must be of type Rmarkdown (.Rmd)
quiet	A logical. Should messages during rendering be surpressed?

Details

This function is merely a simplified version (in terms of possible arguments) of the rendering function from the rmarkdown package. Therefore, we refer to this functions for more details: render. We have included this simplified version in dataMaid in order to help new R users with rendering their output documents as generated by makeDataReport.

See Also

render.

---

setChecks *Set check arguments for makeDataReport*

Description

This function is a tool for easily specifying the checks argument of makeDataReport. Note that all available check function options can be inspected by calling allCheckFunctions().

Usage

```
setChecks(character = defaultCharacterChecks(),
  factor = defaultFactorChecks(), labelled = defaultLabelledChecks(),
  haven_labelled = defaultHavenlabelledChecks(),
  numeric = defaultNumericChecks(), integer = defaultIntegerChecks(),
  logical = defaultLogicalChecks(), Date = defaultDateChecks(),
  all = NULL)
```

Arguments

character	A character vector of function names to be used as checks for character variables. The default options are available by calling defaultCharacterChecks().
factor	A character vector of function names to be used as checks for factor variables. The default options are available by calling defaultFactorChecks().
labelled	A character vector of function names to be used as checks for labelled variables. The default options are available by calling defaultLabelledChecks().
haven_labelled	A character vector of function names to be used as checks for haven_labelled variables. The default options are available by calling defaultHavenlabelledChecks().
numeric	A character vector of function names to be used as checks for numeric variables. The default options are available by calling defaultNumericChecks().

integer	A character vector of function names to be used as checks for integer variables. The default options are available by calling defaultIntegerChecks().
logical	A character vector of function names to be used as checks for logical variables. The default options are available by calling defaultLogicalChecks().
Date	A character vector of function names to be used as checks for Date variables. The default options are available by calling defaultDateChecks().
all	A character vector of function names to be used as checks for all variables. Note that this overrules the choices made for specific variable types by using the other arguments.

Value

A list with one entry for each data class supported by makeDataReport. Each entry then contains a character vector of function names that are to be called as checks for that variable type.

See Also

makeDataReport, allCheckFunctions, defaultCharacterChecks, defaultFactorChecks, defaultLabelledChecks, defaultHavenlabelledChecks, defaultNumericChecks, defaultIntegerChecks, defaultLogicalChecks, defaultDateChecks

Examples

```
#Only identify missing values for characters, logicals and labelled variables:
  setChecks(character = "identifyMissing", factor = "identifyMissing",
     labelled = "identifyMissing")

#Used in a call to makeDataReport():
## Not run:
data(toyData)
makeDataReport(toyData, checks = setChecks(character = "identifyMissing",
   factor = "identifyMissing", labelled = "identifyMissing"), replace = TRUE)

## End(Not run)
```

---

setSummaries *Set summary arguments for makeDataReport*

Description

This function is a tool for easily specifying the summaries argument of makeDataReport. Note that all available summary function options can be inspected by calling allSummaryFunctions().

Usage

```
setSummaries(character = defaultCharacterSummaries(),
  factor = defaultFactorSummaries(),
  labelled = defaultLabelledSummaries(),
  haven_labelled = defaultHavenlabelledSummaries(),
  numeric = defaultNumericSummaries(),
  integer = defaultIntegerSummaries(),
  logical = defaultLogicalSummaries(), Date = defaultDateSummaries(),
  all = NULL)
```

Arguments

character	A character vector of function names to be used as summaries for character variables. The default options are available by calling `defaultCharacterSummaries()`.
factor	A character vector of function names to be used as summaries for factor variables. The default options are available by calling `defaultFactorSummaries()`.
labelled	A character vector of function names to be used as summaries for labelled variables. The default options are available by calling `defaultLabelledSummaries()`.
haven_labelled	A character vector of function names to be used as summaries for haven_labelled variables. The default options are available by calling `defaultHavenlabelledSummaries()`.
numeric	A character vector of function names to be used as summaries for numeric variables. The default options are available by calling `defaultNumericSummaries()`.
integer	A character vector of function names to be used as summaries for integer variables. The default options are available by calling `defaultIntegerSummaries()`.
logical	A character vector of function names to be used as summaries for logical variables. The default options are available by calling `defaultLogicalSummaries()`.
Date	A character vector of function names to be used as summaries for Date variables. The default options are available by calling `defaultDateSummaries()`.
all	A character vector of function names to be used as summaries for all variables. Note that this overrules the choices made for specific variable types by using the other arguments.

Value

A list with one entry for each data class supported by `makeDataReport`. Each entry then contains a character vector of function names that are to be called as summaries for that variable type.

See Also

`makeDataReport`, `allSummaryFunctions`, `defaultCharacterSummaries`, `defaultFactorSummaries`, `defaultLabelledSummaries`, `defaultHavenlabelledSummaries`, `defaultNumericSummaries`, `defaultIntegerSummaries`, `defaultLogicalSummaries`, `defaultDateSummaries`

Examples

```
#Don't include central value (median/mode) summary for numerical and integer
#variables:
  setSummaries(numeric = defaultNumericSummaries(remove = "centralValue"),
    integer = defaultIntegerSummaries(remove = "centralValue"))


#Used in a call to makeDataReport():
## Not run:
data(toyData)
makeDataReport(toyData,
  setSummaries(numeric = defaultNumericSummaries(remove = "centralValue"),
    integer = defaultIntegerSummaries(remove = "centralValue")), replace = TRUE)

## End(Not run)
```

`setVisuals` *Set visual arguments for makeDataReport*

Description

This function is a tool for easily specifying the `visuals` argument of `makeDataReport`. Note that only a single visual function can be provided for each variable type. If more than one is supplied, only the first one is used. The default is to use a single visual function for all variable types (as specified in the argument `all`), but class-specific choices of visual functions can also be used. Note that class-specific arguments overwrites the contents of `all`. Note that all available visual function options can be inspected by calling `allVisualFunctions()`.

Usage

```
setVisuals(character = NULL, factor = NULL, labelled = NULL,
  haven_labelled = NULL, numeric = NULL, integer = NULL,
  logical = NULL, Date = NULL, all = "standardVisual")
```

Arguments

`character`	A function name (character string) to be used as the visual function for character variables. If `NULL` (the default) the argument is ignored and the contents of the `all` argument is used instead.
`factor`	A function name (character string) to be used as the visual function for factor variables. If `NULL` (the default) the argument is ignored and the contents of the `all` argument is used instead.
`labelled`	A function name (character string) to be used as the visual function for labelled variables. If `NULL` (the default) the argument is ignored and the contents of the `all` argument is used instead.

haven_labelled
: A function name (character string) to be used as the visual function for haven_labelled variables. If NULL (the default) the argument is ignored and the contents of the all argument is used instead.

numeric
: A function name (character string) to be used as the visual function for numeric variables. If NULL (the default) the argument is ignored and the contents of the all argument is used instead.

integer
: A function name (character string) to be used as the visual function for integer variables. If NULL (the default) the argument is ignored and the contents of the all argument is used instead.

logical
: A function name (character string) to be used as the visual function for logical variables. If NULL (the default) the argument is ignored and the contents of the all argument is used instead.

Date
: A function name (character string) to be used as the visual function for Date variables. If NULL (the default) the argument is ignored and the contents of the all argument is used instead.

all
: A function name (character string) to be used as the visual function for all variables.

Value

A list with one entry for each data class supported by makeDataReport. Each entry then contains a character string with a function name that is to be called as the visual function for that variable type.

See Also

makeDataReport, allVisualFunctions

Examples

```
#Set visual type to basicVisual for all variable types:
  setVisuals(all = "basicVisual")

#Used in a call to makeDataReport():
## Not run:
data(toyData)
makeDataReport(toyData, visuals = setVisuals(all = "basicVisual"), replace = TRUE)

## End(Not run)
```

`standardVisual` *Produce distribution plots using ggplot from ggplot2.*

Description

Plot the distribution of a variable, depending on its data class, by use of ggplot2. Note that `standardVisual` is a `visualFunction`, compatible with the `visualize` and `makeDataReport` functions.

Usage

```
standardVisual(v, vnam, doEval = TRUE)
```

Arguments

`v`	The variable (vector) to be plotted.
`vnam`	The name of the variable which will appear as the title of the plot.
`doEval`	If TRUE, the plot itself is returned. Otherwise, the function returns a character string containing standalone R code for producing the plot.

Details

For character, factor, logical and (haven_)labelled variables, a barplot is produced. For numeric, integer or Date variables, `standardVisual` produces a histogram instead. Note that for integer and numeric variables, all non-finite (i.e. `NA`, `NaN`, `Inf`) values are removed prior to plotting. For character, Date, factor, (haven_)labelled and logical variables, only `NA` values are removed.

See Also

`visualize`, `basicVisual`

Examples

```
## Not run:
#Save a variable
myVar <- c(1:10)

#Plot a variable
standardVisual(myVar, "MyVar")

#Produce code for plotting a variable
standardVisual(myVar, "MyVar", doEval = FALSE)

## End(Not run)
```

summarize *Summarize a variable/dataset*

Description

Generic shell function that produces a summary of a variable (or for each variable in an entire dataset), given a number of summary functions and depending on its data class.

Usage

```
summarize(v, reportstyleOutput = FALSE, summaries = setSummaries(),
  ...)
```

Arguments

v	The variable (vector) or dataset (data.frame) to be summarized.
reportstyleOutput	Logical indicating whether the output should be formatted for inclusion in the report (escaped matrix) or not. Defaults to not.
summaries	A list of summaries to use on each supported variable type. We recommend using setSummaries for creating this list and refer to the documentation of this function for more details.
...	Additional argument passed to data class specific methods.

Details

Summary functions are supplied using their names (in character strings) in the class-specific argument, e.g. characterSummaries = c("countMissing","uniqueValues") for character variables and similarly for the remaining 7 data classes (factor, Date, labelled, haven_labelled, numeric, integer, logical). Note that an overview of all available summaryFunctions can be obtained by calling allSummaryFunctions.

The default choices of summaryFunctions are available in data class specific functions, e.g. defaultCharacterSummaries() and defaultNumericSummaries(). A complete overview of all default options can be obtained by calling setSummaries()

A user defined summary function can be supplied using its function name. Note however that it should take a vector as argument and return a list on the form list(feature="Feature name",result="The result"). More details on how to construct valid summary functions are found in summaryFunction.

Value

The return value depends on the value of reportstyleOutput.

If reportstyleOutput = FALSE (the default): If v is a varibale, a list of summaryResult objects, one summaryResult for each summary function called on v. If v is a dataset, then summarize() returns a list of lists of summaryResult objects instead; one list for each variable in v.

If `reportstyleOutput = TRUE`: If `v` is a single variable: A matrix with two columns, `feature` and `result` and one row for each summary function that was called. Character strings in this matrix are escaped such that they are ready for Rmarkdown rendering.

If `v` is a full dataset: A list of matrices as described above, one for each variable in the dataset.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (`https://doi.org/10.18637/jss.v090.i06`).

See Also

`setSummaries`, `summaryFunction`, `allSummaryFunctions`, `summaryResult`, `defaultCharacterSumma`
`defaultFactorSummaries`, `defaultLabelledSummaries`, `defaultHavenlabelledSummaries`,
`defaultNumericSummaries`, `defaultIntegerSummaries`, `defaultLogicalSummaries`

Examples

```
#Default summary for a character vector:
   charV <- c("a", "b", "c", "a", "a", NA, "b", "0")
   summarize(charV)

#Inspect default character summary functions:
   defaultCharacterSummaries()

#Define a new summary function and add it to the summary for character vectors:
   countZeros <- function(v, ...) {
     res <- length(which(v == 0))
     summaryResult(list(feature="No. zeros", result = res, value = res))
   }
   summarize(charV,
     summaries = setSummaries(character = defaultCharacterSummaries(add = "countZeros")))

 #Does nothing, as intV is not affected by characterSummaries
   intV <- c(0:10)
   summarize(intV,
     summaries = setSummaries(character = defaultCharacterSummaries(add = "countZeros")))

 #But supplying the argument for integer variables changes the summary:
   summarize(intV, summaries = setSummaries(integer = "countZeros"))

 #Summarize a full dataset:
  data(cars)
  summarize(cars)

 #Summarize a variable and obtain report-style output (formatted for markdown)
  summarize(charV, reportstyleOutput = TRUE)
```

summaryFunction *Create an object of class summaryFunction*

Description

Convert a function, f, into an S3 summaryFunction object. This adds f to the overview list returned by an allSummaryFunctions() call.

Usage

```
summaryFunction(f, description, classes = NULL)
```

Arguments

f	A function. See details and examples below for the exact requirements of this function.
description	A character string describing the summary returned by f. If NULL (the default), the name of f will be used instead.
classes	The classes for which f is intended to be called. If NULL (the default), one of two things happens. If f is not a S3 generic function, the classes attribute of f will be an empty character string. If f is a S3 generic function, an automatic look-up for methods will be conducted, and the classes attribute will then be filled out automatically. Note that the function allClasses (listing all classes used in dataMaid) might be useful.

Details

summaryFunction represents the functions used in summarize and makeDataReport for summarizing the features of variables in a dataset.

An example of defining a new summaryFunction is given below. Note that the minimal requirements for such a function (in order for it to be compatible with summarize() and makeDataReport()) is the following input/output-structure: It must input at least two arguments, namely v (a vector variable) and .... Additional implemented arguments from summarize() and makeDataReport() include maxDecimals, see e.g. the pre-defined summaryFunction minMax for more details about how this arguments should be used. The output must be a list with at least the two entries $feature (a short character string describing what was summarized) and $result (a value or a character string with the result of the summarization). However, if the result of a summaryFunction is furthermore converted to a summaryResult object, a print() method also becomes available for consistent formatting of summaryFunction results.

Note that all available summaryFunctions are listed by the call allSummaryFunctions() and we recommed looking into these function, if more knowledge about summaryFunctions is required.

Value

A function of class summaryFunction which has to attributes, namely classes and description.

See Also

allSummaryFunctions, summarize, makeDataReport, checkResult

Examples

```
#Define a valid summaryFunction that can be called from summarize()
#and makeDataReport(). This function counts how many zero entries a given
#variable has:
 countZeros <- function(v, ...) {
  res <- length(which(v == 0))
  summaryResult(list(feature = "No. zeros", result = res, value = res))
 }

#Convert it to a summaryFunction object. We don't count zeros for
#logical variables, as they have a different meaning here (FALSE):
 countZeros <- summaryFunction(countZeros, description = "Count number of zeros",
                               classes = setdiff(allClasses(), "logical"))

#Call it directly :
 countZeros(c(0, 0, 0, 1:100))

#Call it via summarize():
 data(cars)
 summarize(cars, numericSummaries = c(defaultNumericSummaries(),
   "countZeros"))

#Note that countZeros now appears in a allSummaryFunctions() call:
 allSummaryFunctions()
```

---

summaryResult *Create object of class summaryResult*

Description

Convert a list resulting from the summaries performed in a summaryFunction into a summaryResult object, thereby supplying it with a print() method.

Usage

```
summaryResult(ls)
```

Arguments

ls	A list with entries $feature (a character string describing what summary was obtained), $result (the result of the summary, either a value from the variable, a numeric or a character string) and $value (the result in its most raw format, often identical to the $result input).

Value

A S3 object of class `summaryResult`, identical to the inputted list, `ls`, except for its class attribute.

See Also

`summaryFunction`

---

tableVisual *Produce tables for the makeDataReport visualizations.*

Description

Produce a table of the distribution of a categorical (character, labelled, haven_labelled or factor) variable. Note that `tableVisual` is a `visualFunction`, compatible with the `visualize` and `makeDataReport` functions.

Usage

```
tableVisual(v, vnam, doEval = TRUE)
```

Arguments

`v`	The variable (vector) to be plotted.
`vnam`	The name of the variable.
`doEval`	If TRUE, the table itself is returned. Otherwise, the function returns a character string containing standalone R code for producing the table.

See Also

`visualize`, `basicVisual`, `standardVisual`

Examples

```
## Not run:
#Save a variable
myVar <- c("red", "blue", "red", "red", NA)

#Plot a variable
tableVisual(myVar, "MyVar")

#Produce code for plotting a variable
tableVisual(myVar, "MyVar", doEval = FALSE)

## End(Not run)
```

`testData` *Extended example data to test the features of dataMaid*

Description

A dataset of constructed data used as test bed when using `dataMaid` for identifying potential errors in a dataset.

Usage

```
testData
```

Format

A data frame with 15 rows and 14 variables.

charVar A character vector with a single missing observation.

factorVar A factor vector with a miscoded missing observation, `999`.

numVar A numeric vector

intVar An integer vector

boolVar A logical vector with three missing observations.

keyVar A character vector with unique codes for each observation.

emptyVar A numeric vector where all entries are identical.

numOutlierVar A numeric vector with a possible outlier (`100`).

smartNumVar A numeric vector that takes only two different values.

cprVar A character vector with levels in the format of Danish CPR numbers (social security numbers).

cprKeyVar A character vector with levels in the format of Danish CPR numbers (social security numbers) with unique levels for each observation.

miscodedMissingVar A character vector with levels corresponding to various miscoded (non-`NA`) misssing codes.

misclassifiedNumVar A misclassified factor variable, where every level is a number and a many (12) different levels are in use.

dateVar A Date vector.

labelledVar A labelled vector with two missing observations.

Source

Artificial data

Examples

```
data(testData)
```

toyData *Small example data to show the features of dataMaid*

Description

An artificial dataset, intended for presenting the key features of `dataMaid`, which is a toolset for identifying potential errors in a dataset.

Usage

```
toyData
```

Format

A `data.frame` with 15 rows and 6 variables.

pill A factor variable with two levels (`"red"` and `"blue"`) and a few (correctly coded) missing observations. This represents the colour of a pill.

events A numeric variable with one obvious outlier value (`82`), two miscoded missing values (`999` and `NaN`) and a few correctly coded missing values. The number of previous events.

region A factor variable where two of the levels (`"other"` and `"OTHER"` are the same word with different case settings. Moreover, the variable includes a Stata-style miscoded missing value (`". "`). Used to represent geographical regions or treatment centers..

change A numeric variable (random draws from a standard normal distribution). Representing a change in a measured variable.

id A factor variable with unique codes for each observation (a character string with a number between 1 and 15), i.e. a key variable.

spotifysong A factor variable that has the same level (`"Irrelevant"`) for all observations, i.e. a empty variable. The latest song played on Spotify.

Source

Artificial data

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (`https://doi.org/10.18637/jss.v090.i06`).

Examples

```
data(toyData)
```

uniqueValues *summaryFunction for unique values*

Description

A `summaryFunction` type function, intended to be called from `summarize` to be called from `summarize`, which counts the number of unique (excluding `NA`s) values in a variable.

Usage

```
uniqueValues(v, ...)
```

Arguments

`v`	A variable (vector).
`...`	Not in use.

Value

An object of class `summaryResult` with the following entries: `$feature` ("No. unique values") and `$result` (the number of unique values in `v`).

See Also

`summaryFunction`, `summarize`, `summaryResult`, `allSummaryFunctions`

Examples

```
uniqueValues(c(1:3, rep(NA, 10), Inf, NaN))
```

variableType *Summary function for original class*

Description

A `summaryFunction` type function, intended to be called from `summarize`, which finds the original class of a variable. This is just the class for all objects but those of class `smartNum`.

Usage

```
variableType(v, ...)
```

Arguments

`v`	A variable (vector).
`...`	Not in use.

Value

An object of class `summaryResult` with the following entries: `$feature` ("Variable type"), `$result` (the (original) class of `v`) and `$value` (identical to `$result`).

See Also

`summarize`

Examples

```
#For standard variables:
  varX <- c(rep(c(1,2,3), each=10))
  class(varX)
  variableType(varX)

#For smartNum variables:
  smartX <- dataMaid:::smartNum(varX)
  class(smartX)
  variableType(smartX)
```

`visualFunction` *Create an object of class visualFunction*

Description

Convert a function, `f`, into an S3 `visualFunction` object. This adds `f` to the overview list returned by an `allVisualFunctions()` call.

Usage

```
visualFunction(f, description, classes = NULL)
```

Arguments

`f`	A function. See details and examples below for the exact requirements of this function.
`description`	A character string describing the visualization returned by `f`. If `NULL` (the default), the name of `f` will be used instead.
`classes`	The classes for which `f` is intended to be called. If `NULL` (the default), one of two things happens. If `f` is not a S3 generic function, the `classes` attribute of `f` will be an empty character string. If `f` is a S3 generic function, an automatic look-up for methods will be conducted, and the `classes` attribute will then be filled out automatically. Note that the function `allClasses` (listing all classes used in `dataMaid`) might be useful.

Details

visualFunction represents the functions used in visualize and makeDataReport for plotting the distributions of the variables in a dataset.

An example of defining a new visualFunction is given below. Note that the minimal requirements for such a function (in order for it to be compatible with visualize() and makeDataReport()) is the following input/output-structure: It must input exactly the following three arguments, namely v (a vector variable), vnam (a character string with the name of the variable) and doEval (a logical). The last argument is supposed to control whether the function produces a plot in the graphic device (if doEval = TRUE) or instead returns a character string including R code for generating such a plot. In the latter setting, the code must be stand-alone, that is, it cannot depend on object available in an environment. In practice, this will typically imply that the data variable is included in the code snip. It is not strictly necessary to implement the doEval = TRUE setting for the visualFunction to be compatible with makeDataReport, but we recommend doing it anyway such that the function can also be used interactively.

Note that all available visualFunctions are listed by the call allVisualFunctions() and we recommed looking into these function, if more knowledge about visualFunctions is required.

Value

A function of class visualFunction which has to attributes, namely classes and description.

See Also

allVisualFunctions, visualize, makeDataReport

Examples

```
#Defining a new visualFunction:
 mosaicVisual <- function(v, vnam, doEval) {
   thisCall <- call("mosaicplot", table(v), main = vnam, xlab = "")
   if (doEval) {
    return(eval(thisCall))
   } else return(deparse(thisCall))
 }
 mosaicVisual <- visualFunction(mosaicVisual, description = "Mosaicplots from graphics",
                                classes = allClasses())

#mosaicVisual is now included in a allVisualFunctions() call:
 allVisualFunctions()

#Create a mosaic plot:
 ABCvar <- c(rep("a", 10), rep("b", 20), rep("c", 5))
 mosaicVisual(ABCvar, "ABCvar", TRUE)

#Create a character string with the code for a mosaic plot:
 mosaicVisual(ABCvar, "ABCVar", FALSE)

#Extract or set description of a visualFunction:
 description(mosaicVisual)
```

```
description(mosaicVisual) <- "A cubist version of a pie chart"
description(mosaicVisual)
```

`visualize` *Produce distribution plots*

Description

Generic shell function that calls a plotting function in order to produce a marginal distribution plot for a variable (or for each variable in a dataset). What type of plot is made might depend on the data class of the variable.

Usage

```
visualize(v, vnam = NULL, visuals = setVisuals(), doEval = TRUE, ...)
```

Arguments

`v`	The variable (vector) or dataset (data.frame) which is to be plotted.
`vnam`	The name of the variable. This name might be printed on the plots, depending on the choice of plotting function. If not supplied, it will default to the name of `v`.
`visuals`	A list of visual functions to use on each supported variable type. We recommend using `setVisuals` for creating this list and refer to the documentation of this function for more details. This function allows for choosing variable-type dependent visuals. However, if `visualize()` is called on a full dataset, all visualizations must be of the same type and therefore, the `all` argument of `setVisuals` is used.
`doEval`	A logical. If `TRUE` (the default), `visualize` has the side effect of producing a plot (or multiple plots, if `v` is a data.frame). Otherwise, visualize returns a character string containing R-code for producing the plot (or, when `v` is a data.frame, a list of such character strings).
`...`	Additional arguments used for class-specific choices of visual functions (see *details*).

Details

Visual functions can be supplied using their names (in character strings) using `setVisuals`. Note that only a single visual function is allowed for each variable class. The default visual settings can be inspected by calling `setVisuals()`. An overview of all available `visualFunctions` can be obtained by calling `allVisualFunctions`.

A user defined visual function can be supplied using its function name. Details on how to construct valid visual functions are found in `visualFunction`.

References

Petersen AH, Ekstrøm CT (2019). "dataMaid: Your Assistant for Documenting Supervised Data Quality Screening in R." _Journal of Statistical Software_, *90*(6), 1-38. doi: 10.18637/jss.v090.i06 (https://doi.org/10.18637/jss.v090.i06).

See Also

setVisuals, allVisualFunctions, standardVisual, basicVisual

Examples

```
#Standard use: Return standalone code for plotting a function:
  visualize(c(1:10), "Variable 1", doEval = FALSE)

#Define a new visualization function and call it using visualize either
#using allVisual or a class specific argument:
  mosaicVisual <- function(v, vnam, doEval) {
    thisCall <- call("mosaicplot", table(v), main = vnam, xlab = "")
    if (doEval) {
      return(eval(thisCall))
    } else return(deparse(thisCall))
  }
  mosaicVisual <- visualFunction(mosaicVisual,
                                 description = "Mosaicplots from graphics",
                                 classes = allClasses())

 #Inspect all options for visualFunctions:
 allVisualFunctions()


## Not run:
  #set mosaicVisual for all variable types:
  visualize(c("1", "1", "1", "2", "2", "a"), "My variable",
      visuals = setVisuals(all = "mosaicVisual"))

  #set mosaicVisual only for character variables:
  visualize(c("1", "1", "1", "2", "2", "a"), "My variable",
     visuals = setVisuals(character = "mosaicVisual"))

  #this will use standardVisual, as our variable is not numeric:
  visualize(c("1", "1", "1", "2", "2", "a"), "My variable",
      visuals = setVisuals(numeric = "mosaicVisual"))

## End(Not run)

  #return code for a mosaic plot
  visualize(c("1", "1", "1", "2", "2", "a"), "My variable",
      allVisuals = "mosaicVisual", doEval=FALSE)

## Not run:
#Produce multiple plots easily by calling visualize on a full dataset:
  data(testData)
```

```
  testData2 <- testData[, c("charVar", "factorVar", "numVar", "intVar")]
  visualize(testData2)

#When using visualize on a dataset, datatype specific arguments have no
#influence:
  visualize(testData2, setVisuals(character = "basicVisual",
      factor = "basicVisual"))

#But we can still use the "all" argument in setVisuals:
  visualize(testData2, visuals =  setVisuals(all = "basicVisual"))

## End(Not run)
```