Package 'clinUtils'

April 23, 2024

Type Package

Title General Utility Functions for Analysis of Clinical Data

Version 0.1.5

Date 2024-04-22

Description Utility functions to facilitate the import,
the reporting and analysis of clinical data.
Example datasets in 'SDTM' and 'ADaM' format, containing a subset of patients/domains
from the 'CDISC Pilot 01 study' are also available as R datasets to demonstrate
the package functionalities.

Imports crosstalk, data.table, DT, haven, htmlwidgets, knitr, plyr,
tools, utils, viridisLite

Suggests tibble, ggplot2, plotly, htmltools, pander, rmarkdown,
testthat, flextable

SystemRequirements pandoc (for export clin DT to a file - inclusion of
list of interactive plots/objects with knitr)

URL https://github.com/openanalytics/clinUtils

BugReports https://github.com/openanalytics/clinUtils/issues

License MIT + file LICENSE

RoxygenNote 7.3.1

VignetteBuilder knitr

LazyData true

NeedsCompilation no

Author Laure Cougnaud [aut, cre],
Michela Pasetto [aut],
Arne De Roeck [rev] (tests),
Open Analytics [cph]

Maintainer Laure Cougnaud <laure.cougnaud@openanalytics.eu>

Repository CRAN

Date/Publication 2024-04-23 20:50:31 UTC

R topics documented:

checkVarInData *Check if specified variables are in the data. If they are not, they are removed from specified variables and a message is printed.*

Description

Check if specified variables are in the data. If they are not, they are removed from specified variables and a message is printed.

Usage

```
checkVarInData(var, data, label)
```

Arguments

var	Character vector with variables.
data	Data.frame with data.
label	String with label used in message.

Value

var present in data, NULL if empty

clinColors *Colors of 'clinUtils'*

Description

Default color palette is the color-blind viridis palette. See documentation of viridis.

Usage

```
clinColors(n, alpha = 1, begin = 0, end = 1, direction = 1, option = "D")
```

Arguments

n	The number of colors (≥ 1) to be in the palette.
alpha	The alpha transparency, a number in [0,1], see argument alpha in hsv.
begin	The (corrected) hue in [0,1] at which the color map begins.
end	The (corrected) hue in [0,1] at which the color map ends.
direction	Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.
option	see similar parameter in viridis

Value

The viridis function.

clinLinetypes *Linetypes of 'clinUtils'*

Description

A set of default linetypes are available as a vector.

Usage

```
clinLinetypes
```

Format

An object of class `character` of length 6.

Value

A character vector of linetypes.

clinShapes *Shapes of 'clinUtils'*

Description

A set of default shapes are available as named vector.

Usage

```
clinShapes
```

Format

An object of class `integer` of length 24.

Value

A numeric vector of shapes.

clinShapesText *Shapes of 'clinUtils' as text*

Description

A set of default shapes are available as character vector.

Usage

```
clinShapesText
```

Format

An object of class `character` of length 24.

Value

A character vector of shapes.

clinUtils-palette *Parameters for all palette functions for clinical visualizations.*

Description

Parameters for all palette functions for clinical visualizations.

Arguments

`n`	Integer of length 1, number of elements in palette.
`x`	Vector with elements used for palette. If factor, the levels are used, otherwise the unique elements of the vector. Missing values are automatically removed, excepted if `includeNA` is set to TRUE.
`includeNA`	Logical (FALSE by default), should NA elements be retained in the palette in case `x` is specified?

Value

Specific palettes used in `clinUtils`.

colorPaletteNRIND — *Color palette for a standard CDISC Normal/Reference Range Indicator.*

Description

Color palette for a standard CDISC Normal/Reference Range Indicator.

Usage

```
colorPaletteNRIND
```

Format

A named character vector with color for typical Normal Reference Range Indicator variable:

- "LOW": orange
- "NORMAL": green4
- "HIGH": orange
- "ABNORMAL": red
- "UNKNOWN" or 'NA': grey
- "NA": grey

compareDiff — *Get differences between two data.frames*

Description

Get differences between two data.frames

Usage

```
compareDiff(
  newData,
  oldData,
  referenceVars = intersect(colnames(newData), colnames(oldData)),
  changeableVars = NULL
)
```

Arguments

`newData`	data.frame object representing the new data
`oldData`	data.frame object representing the old data
`referenceVars`	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in `newData` and `oldData` are considered.
`changeableVars`	character vector of the columns in the data for which you want to assess the change, e.g. variables that might have changed from the old to the new data. If not specified, only 'Addition' and 'Removal' are detected.

Value

Object of class 'diff.data', i.e. a data.frame with columns:

- 'Comparison type': type of difference between the old and new data, either:
  - 'Change': records present both in new and old data, based on the reference variables, but with difference(s) in changeable vars
  - 'Addition': records with reference variables present in new but not in old data
  - 'Removal': records with reference variables present in old but not in new data
- 'Version': 'Previous' or 'Current' depending if record represents content from old or new data respectively
- `referenceVars`
- `changeableVars`

Identification of the differences between datasets

To identify the differences between datasets, the following steps are followed:

1. removal of records identical between the old and new dataset (will be considered as 'Identical' later on)
2. records with a reference value present in the old dataset but not in the new dataset are considered 'Removal'
3. records with a reference value present in the new dataset but not in the old dataset are considered 'Addition'
4. records with reference value present both in the new and old dataset, **after filtering of identical records** and with difference in the changeable variables are considered 'Change'

Author(s)

Laure Cougnaud

compareTables *Compare tables*

Description

Compare tables

Usage

```
compareTables(
  newData,
  oldData,
  referenceVars = intersect(colnames(newData), colnames(oldData)),
  changeableVars = NULL,
  outputType = c("table-comparison", "newData-diff", "oldData-diff",
    "table-comparison-interactive", "newData-diff-interactive",
    "oldData-diff-interactive"),
  ...
)
```

Arguments

newData	data.frame object representing the new data
oldData	data.frame object representing the old data
referenceVars	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in newData and oldData are considered.
changeableVars	character vector of the columns in the data for which you want to assess the change, e.g. variables that might have changed from the old to the new data. If not specified, only 'Addition' and 'Removal' are detected.
outputType	String describing which output should be returned, (multiple are possible), either: • 'table-comparison': data.frame containing difference between two datasets, see 'output' of compareDiff function. • 'table-comparison-interactive': datatable object with differences between the two datasets, see 'output' of exportDiffData. • 'newData-diff' or 'oldData-diff': data.frame with new/old data respectively, containing the information if each record differs in the old/new datasets respectively. See output of mergeDiffWithData. • 'newData-diff-interactive' or 'oldData-diff-interactive': datatable with new/old data respectively, containing the information if each record differs in the old/new datasets respectively. See output of exportDiffData.
...	Any parameters passed to the exportDiffData function. These are only used if 'table-comparison-interactive' is specified in outputType.

Value

One of the output types specified in `outputType`. By default, all outputs are returned. If multiple output types are specified, a list of those are returned (named by output type).

Identification of the differences between datasets

To identify the differences between datasets, the following steps are followed:

1. removal of records identical between the old and new dataset (will be considered as 'Identical' later on)
2. records with a reference value present in the old dataset but not in the new dataset are considered 'Removal'
3. records with a reference value present in the new dataset but not in the old dataset are considered 'Addition'
4. records with reference value present both in the new and old dataset, **after filtering of identical records** and with difference in the changeable variables are considered 'Change'

Author(s)

Laure Cougnaud, Michela Pasetto

Examples

```
## Example 1
# In this case the referenceVar 'a' is the same
# the comparison highlights only as change in the variables 'c' and 'd'
newData <- data.frame(
"a" = c(1, 2, 3, 4),
"b" = c(5, 6, 7, 8),
"c" = rep(1, 4),
"d" = rep(2, 4)
)
oldData <- data.frame(
"a" = c(1, 2, 3, 4),
"b" = c(3, 4, 7, 8),
"c" = rep(2, 4),
"d" = rep(1, 4)
)
compareTables(
newData = newData,
oldData = oldData,
referenceVars = "a",
changeableVars = c("c", "d")
)

## Example 2
# In this case the referenceVar 'a' changes in the last two rows
# the comparison highlights as change the second and third rows in the variables 'c' and 'd'
# whereas the last rows are additions/removals with respect to the reference 'a'
newData <- data.frame(
```

```
"a" = c(7, 1, 2, 3, 4),
"b" = c(2, 1, 6, 7, 8),
"c" = rep(1, 5),
"d" = rep(2, 5)
)
oldData <- data.frame(
"a" = c(7, 1, 2, 5, 6),
"b" = c(2, 3, 4, 7, 8),
"c" = c(1, rep(2, 4)),
"d" = c(2, rep(1, 4))
)
compareTables(
newData = newData,
oldData = oldData,
referenceVars = "a",
changeableVars = c("c", "d")
)

## Example 3
# In this case the referenceVar 'a' is the same
# also the variable 'c' is the same and it's the only changeable var evaluated
newData <- data.frame(
"a" = c(1, 2, 3, 4),
"b" = c(5, 6, 7, 8),
"c" = rep(1, 4),
"d" = rep(2, 4)
)
oldData <- data.frame(
"a" = c(1, 2, 3, 4),
"b" = c(3, 4, 7, 8),
"c" = rep(1, 4),
"d" = rep(1, 4)
)
compareTables(
newData = newData,
oldData = oldData,
referenceVars = "a",
changeableVars = "c"
)

## Not run: # due to time constraint in CRAN

## In case only a specific output should be returned:
newData <- data.frame(
"a" = c(7, 1, 2, 3, 4),
"b" = c(2, 1, 6, 7, 8),
"c" = rep(1, 5),
"d" = rep(2, 5)
)
oldData <- data.frame(
"a" = c(7, 1, 2, 5, 6),
"b" = c(2, 3, 4, 7, 8),
"c" = c(1, rep(2, 4)),
```

```
"d" = c(2, rep(1, 4))
)

# get only the differences between datasets:

# as a data.frame
compareTables(newData = newData, oldData = oldData,
referenceVars = "a", changeableVars = c("c", "d"),
outputType = "table-comparison")

# as an interactive DataTable
compareTables(newData = newData, oldData = oldData,
referenceVars = "a", changeableVars = c("c", "d"),
outputType = "table-comparison-interactive"
)
# only the new data
compareTables(
    newData = newData, oldData = oldData,
referenceVars = "a", changeableVars = c("c", "d"),
outputType = "newData-diff"
)
# only the new data in interactive mode
compareTables(
    newData = newData, oldData = oldData,
    referenceVars = "a", changeableVars = c("c", "d"),
    outputType = "newData-diff-interactive"
)
# only the new data in static and interactive mode
compareTables(
    newData = newData, oldData = oldData,
    referenceVars = "a", changeableVars = c("c", "d"),
    outputType = c("newData-diff", "newData-diff-interactive")
)
# only the old data
compareTables(newData = newData, oldData = oldData,
referenceVars = "a", changeableVars = c("c", "d"),
outputType = "oldData-diff"
)
# only the old data in interactive mode
compareTables(
    newData = newData, oldData = oldData,
    referenceVars = "a", changeableVars = c("c", "d"),
    outputType = "oldData-diff-interactive"
)
# only the old data in static and interactive mode
compareTables(
    newData = newData, oldData = oldData,
    referenceVars = "a", changeableVars = c("c", "d"),
    outputType = c("oldData-diff", "oldData-diff-interactive")
)


## End(Not run)
```

```
## no changeable vars

newData <- data.frame(
"a" = c(7, 1, 2, 3, 4),
"b" = c(2, 1, 6, 7, 8),
"c" = rep(1, 5),
"d" = rep(2, 5)
)
oldData <- data.frame(
"a" = c(7, 1, 2, 5, 6),
"b" = c(2, 3, 4, 7, 8),
"c" = c(1, rep(2, 4)),
"d" = c(2, rep(1, 4))
)

compareTables(newData = newData, oldData = oldData,
referenceVars = "a"
)

## duplicated records

# in case there are multiple records for the same reference variables,
# identical records are flagged as 'Identity' and reported in the table
# reporting differences; and the different record are flagged as 'Change', 'Addition' or 'Removal'
newData <- data.frame(
"a" = c(7, 7),
"b" = c(1, 2),
"c" = c(1, 2),
"d" = c(2, 3)
)
oldData <- data.frame(
"a" = c(7, 7, 7),
"b" = c(3, 4, 5),
"c" = c(1, 3, 5),
"d" = c(2, 4, 6)
)
compareTables(
newData = newData, oldData = oldData,
referenceVars = "a", changeableVars = c("c", "d"),
)

## with labels in the interactive format, see ? getClinDT

## Not run: # due to time constraint in CRAN

newData <- data.frame(
"a" = c(7, 1, 2, 3, 4),
"b" = c(2, 1, 6, 7, 8),
"c" = rep(1, 5),
"d" = rep(2, 5)
)
oldData <- data.frame(
"a" = c(7, 1, 2, 5, 6),
```

```
"b" = c(2, 3, 4, 7, 8),
"c" = c(1, rep(2, 4)),
"d" = c(2, rep(1, 4))
)
compareTables(
newData = newData,
oldData = oldData,
referenceVars = "a",
changeableVars = c("c", "d"),
# parameters passed to datatable
colnames = c(
"My reference variable" = "a",
"Changeable variable c" = "c",
"Changeable variable d" = "d"
)
)


## End(Not run)
```

`comparisonTables-common-args`
General parameters used for the comparison table functionality

---

Description

General parameters used for the comparison table functionality

Arguments

`newData`	data.frame object representing the new data
`oldData`	data.frame object representing the old data
`referenceVars`	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in `newData` and `oldData` are considered.
`changeableVars`	character vector of the columns in the data for which you want to assess the change, e.g. variables that might have changed from the old to the new data. If not specified, only 'Addition' and 'Removal' are detected.
`diffData`	Object of class 'diff.data' containing differences between datasets, as returned by the `compareDiff` function.
`outputType`	String describing which output should be returned, (multiple are possible), either: • 'table-comparison': data.frame containing difference between two datasets, see 'output' of `compareDiff` function.

- 'table-comparison-interactive': `datatable` object with differences between the two datasets, see 'output' of `exportDiffData`.
- 'newData-diff' or 'oldData-diff': data.frame with new/old data respectively, containing the information if each record differs in the old/new datasets respectively. See output of `mergeDiffWithData`.
- 'newData-diff-interactive' or 'oldData-diff-interactive': `datatable` with new/old data respectively, containing the information if each record differs in the old/new datasets respectively. See output of `exportDiffData`.

Value

The comparison of the two input tables.

`convertToDatatable` *Convert to data.table*

Description

Convert a data frame into a `data.table` object.

Usage

```
convertToDatatable(data)
```

Arguments

`data`	A data.frame

Value

A `data.table` object.

`convertToDateTime` *Convert character vector to date/time object*

Description

Convert character vector to date/time object

Usage

```
convertToDateTime(
  x,
  format = c("%Y-%m-%dT%H:%M", "%Y-%m-%d"),
  colName = NULL,
  verbose = TRUE
)
```

Arguments

x	character vector to convert to date/time
format	string with possible format(s) of the input date/time in the ADaM dataset. If multiple are specified, each format is tested successively, until at least one element in the input vector is converted with the specified format (non missing, following the approach described in the format parameter of the strptime function). See the 'Details' section of the help of the function, for more information about this format.
colName	string with name of column, used in message (if any).
verbose	logical, if TRUE (by default) progress messages are printed during execution

Value

Vector of class POSIXct

Author(s)

Laure Cougnaud

---

dataADaMCDISCP01 *Example of ADaM datasets from the CDISC original Pilot 01 study*

Description

This contains a subset of the CDISC Pilot 01 study dataset for:

- a selected subset of subjects
- a selected subset of domains:
  - subject-level ('adsl')
  - adverse event ('adae')
  - laboratory chemistry data('adlbc')
  - vital signs ('advs')
  - concomitant medications ('adcm')
  - efficacy:
    - ADAS-COG Data ('adqsadas'), containing one of the primary endpoint: ADAS-Cog (11)
      (Alzheimer's Disease Assessment Scale - Cognitive Subscale
    - CIBIC+ questionnaire data ('adqscibc'), containing one of the primary endpoint: CIBIC+
      (Video-referenced Clinician's Interview-based Impression of Change)
    - NPI-X Item data ('adqsnpix'), containing the secondary endpoint: NPI-X
      (Mean Revised Neuropsychiatric Inventory)
  - pharmacokinetic parameters ('adpp')
    Please note that this dataset contains different sets of subjects than the other example datasets.

This dataset was created following the ADaM Version 2.0 standard. This dataset contains the 'Modified and augmented version of cdiscpilot01' dataset.

Format

List of data.frames containing the ADaM dataset for each selected domain.
Labels for the different variables across datasets is available via the `labelVars` attribute.

Author(s)

Laure Cougnaud

Source

Original (and entire) datasets are available in:
https://github.com/phuse-org/phuse-scripts/tree/master/data/adam/cdisc See in particular the *define.xml* file for further description of the datasets and variables name.

See Also

loadDataADaMSDTM

---

dataSDTMCDISCP01 *Example of SDTM datasets from the CDISC original Pilot 01 study*

Description

This contains a subset of the CDISC original Pilot 01 study dataset for:

- a selected subset of subjects
- a selected subset of domains:
  - adverse event ('ae')
  - concomitant medications ('cm')
  - disposition ('ds')
  - demographics ('dm')
  - treatment exposure ('ex')
  - laboratory ('lb')
  - medical history ('mh')
  - questionnaire ('qs') - only:
    - ADAS-Cog (11) primary endpoint (`QSTESTCD == 'ACTOT'`)
    - CIBIC+ primary endpoint (`QSTESTCD == 'CIBIC'`)
    - NPI-X Item secondary endpoint (`QSTESTCD == 'NPTOT'`)
  - demographics supplemental dataset ('suppdm')
  - subject visits ('sv')
  - vital signs ('vs')

This dataset was created following the SDTM Version 2 standard.

Format

List of data.frames containing the SDTM dataset for each selected domain.
Labels for the different variables across datasets is available via the `labelVars` attribute.

Author(s)

Laure Cougnaud

Source

Original (and entire) datasets are available in:
https://github.com/phuse-org/phuse-scripts/tree/master/data/sdtm/cdiscpilot01 See in particular the *define.xml* file for further description of the datasets and variables name.

See Also

loadDataADaMSDTM

`exportDiffData` *Export the 'diff.data' object from* `compareDiff` *to a user-friendly format*

Description

Export the 'diff.data' object from `compareDiff` to a user-friendly format

Usage

```
exportDiffData(
  diffData,
  newDataDiff,
  oldDataDiff,
  referenceVars = attr(diffData, "referenceVars"),
  changeableVars = attr(diffData, "changeableVars"),
  to = "DT",
  ...
)
```

Arguments

`diffData`	Object of class 'diff.data' containing differences between datasets, as returned by the `compareDiff` function.
`newDataDiff`	data.frame with new data with differences as returned by the `mergeDiffWithData`. The data set contains the new data with the information if each record differs in the new dataset.

`oldDataDiff`	data.frame with old data with differences as returned by the `mergeDiffWithData`. The data set contains the old data with the information if each record differs in the old dataset.
`referenceVars`	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in `newData` and `oldData` are considered.
`changeableVars`	character vector of the columns in the data for which you want to assess the change, e.g. variables that might have changed from the old to the new data. If not specified, only 'Addition' and 'Removal' are detected.
`to`	String with export format, currently only: `DT` is available to export to a `datatable` object.
`...`	Extra parameters besides 'data' and 'nonVisibleVars', currently passed to the `getClinDT` function.

Value

Depending on the `to` parameter:

- 'DT': a `datatable` with the difference between datasets, with:
  - highlighting depending on the difference between datasets:
    - 'Addition' in green
    - 'Removal' in yellow
    - 'Change' in lightblue
    - 'Identical' are not highlighted
  - records only present in the old dataset are displayed in italic

---

`formatDetailsComparison`

Format details comparison

Description

Format details comparison

Usage

```
formatDetailsComparison(
  diffData,
  referenceVars = attr(diffData, "referenceVars"),
  changeableVars = attr(diffData, "changeableVars")
)
```

Arguments

diffData	Object of class 'diff.data' containing differences between datasets, as returned by the compareDiff function.
referenceVars	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in newData and oldData are considered.
changeableVars	character vector of the columns in the data for which you want to assess the change, e.g. variables that might have changed from the old to the new data. If not specified, only 'Addition' and 'Removal' are detected.

Value

diffData with extra columns: '[].diff' for the referenceVars and changeableVars columns, and attributes: 'colsDiff' as a named vector with mapping with input variables (names) and corresponding diff variables.

Author(s)

Laure Cougnaud

---

formatDTBarVar *Format a variable in a datatable as a barplot.*

Description

Format a variable in a datatable as a barplot.

Usage

```
formatDTBarVar(
  tableDT,
  data,
  barVar = NULL,
  barColorThr = NULL,
  barRange = NULL,
  getCol = function(x) x
)
```

Arguments

tableDT	datatable object
data	Data.frame with content of tableDT.
barVar	Character vector with numeric variable of data which should be represented as bar in the table.

barColorThr	Numeric vector with threshold to consider to color the bar, either: • a numeric vector of length 1, same threshold for all bars • named vector with threshold for each bar, named with the variable in barVar
barRange	(optional) range for the bars, either: • a numeric vector of length 2, same range for all bars • list with range for each bar, named with the variable in barVar If not specified, the range of each barVar variable in data is used.
getCol	Function, which for an index of a column in data returns the index of the column to be passed to formatStyle

Value

Updated tableDT

Author(s)

Laure Cougnaud

---

formatLabel *Concatenate and format text strings to a label*

Description

This function concatenates and formats text strings to a label e.g to use for chunk and table/figures

Usage

```
formatLabel(...)
```

Arguments

...	string(s) to be concatenated to form label or data.frame with only one row. If an unique data.frame is specified, the different columns are collapsed to form one label.

Value

String with chunk label

Author(s)

Laure Cougnaud

formatLabelChunk *Concatenate and format text strings to a chunk label*

Description

Concatenate and format text strings to a chunk label

Usage

```
formatLabelChunk(...)
```

Arguments

...	string to be concatenated to form chunk label

Value

String with chunk label

Author(s)

Laure Cougnaud

formatLongLabel *Format a variable with long labels*

Description

This function formats a variable with long labels by wrapping its elements into multiple lines.

Usage

```
formatLongLabel(x, width = 20)
```

Arguments

x	character vector with labels to format
width	target maximum size. Note: a word longer that this width won't be split (see strwrap).

Value

Vector with formatted labels

Author(s)

Laure Cougnaud longLabel <- "This is a very long description of the variable in the dataset" cat(longLabel) cat(formatLongLabel(longLabel))

formatTableLabel *Concatenate and format text strings to a label of a table*

Description

This function concatenates and formats text strings to a label of a table for bookdown package

Usage

```
formatTableLabel(...)
```

Arguments

...	string to be concatenated to form label

Value

String with chunk label

Author(s)

Laure Cougnaud

formatVarForPlotLabel *Format parameter variable to be displayed in the labels of a plot*

Description

The following workflow is used:

1. format the variable as a factor
2. wrap it across multiple lines if needed
3. sort (its levels) according to a grouping variable

Usage

```
formatVarForPlotLabel(
  data,
  paramVar = NULL,
  paramGroupVar = NULL,
  revert = FALSE,
  width = 20
)
```

Arguments

data	data.frame with data
paramVar	string, variable of data with parameter
paramGroupVar	(optional) character vector with variable(s) of data with grouping. If specified, the parameters will be grouped by this(these) variable(s) in the y-axis.
revert	logical, if TRUE revert the order of the levels of the variable
width	max number of characters in the paramVar parameter.

Value

Vector with re-formatted paramVar, NULL if empty

Author(s)

Laure Cougnaud library(ggplot2) data(dataADaMCDISCP01) dataAE <- dataADaMCDISCP01$ADAE

by default, groups are sorted alphabetically in ggplot2 (from bottom to top for an histogram) ggplot(data = dataAE, aes(y = AEDECOD, fill = AEBODSYS)) + geom_histogram(stat="count")

by default: labels are set to a new line if more than 20 characters: dataAE$AEDECOD <- formatVarForPlotLabel(data = dataAE, paramVar = "AEDECOD") levels(dataAE$AEDECOD) ggplot(data = dataAE, aes(y = AEDECOD, fill = AEBODSYS)) + geom_histogram(stat="count")

revert order of the variable dataAE$AEDECOD <- formatVarForPlotLabel(data = dataAE, paramVar = "AEDECOD", revert = TRUE) levels(dataAE$AEDECOD) ggplot(data = dataAE, aes(y = AEDECOD, fill = AEBODSYS)) + geom_histogram(stat="count")

group based on body system dataAE$AEDECOD <- formatVarForPlotLabel(data = dataAE, paramVar = "AEDECOD", paramGroupVar = "AEBODSYS") ggplot(data = dataAE, aes(y = AEDECOD, fill = AEBODSYS)) + geom_histogram(stat="count")

---

getClinDT *Create an interactive table to display clinical data*

Description

This function converts a data.frame from R into a datatable object with sensitive defaults. Extra functionalities are available to:

- have columns or cells of interest that are collapsible/expandable (see expandVar/expandIdx)
- group rows based on a variable (see rowGroupVar)
- display a variable as barplot (with specified range of threshold) (see barVar)
- hide variable(s) (see nonVisibleVar)

Usage

```
getClinDT(
  data,
  nonVisibleVar = NULL,
  nonVisible = NULL,
  percVar = NULL,
  barVar = NULL,
  barColorThr = NULL,
  barRange = NULL,
  filter = "top",
  searchBox = FALSE,
  pageLength,
  fixedColumns = NULL,
  columnsWidth = NULL,
  options = list(),
  expandVar = NULL,
  expandIdx = NULL,
  escape = TRUE,
  rowGroup = NULL,
  rowGroupVar = NULL,
  vAlign = "top",
  callback = NULL,
  buttons = getClinDTButtons(),
  scrollX = TRUE,
  file = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`data`	Data.frame, matrix or `SharedData` object with input data for the table.
`nonVisibleVar`	Character vector with column(s) in `data` to hide in the output table (column is hidden). The column(s) also get the extra attribute: `className = 'noVis'`, to ensure they are not displayed in the button to show/hide column(s).
`nonVisible`	This parameter is deprecated, use the new interface with the `nonVisibleVar` parameter. Numeric vector with column(s) in `data` to not display in the output table (column is hidden), in **Javascript unit: first column is 0**, second column is 1, ...
`percVar`	Character vector with percentage columns. These columns should contain the percentage from 0 to 1. The content of these colunms will be rounded to 2 digits.
`barVar`	Character vector with numeric variable of `data` which should be represented as bar in the table.
`barColorThr`	Numeric vector with threshold to consider to color the bar, either: • a numeric vector of length 1, same threshold for all bars

- named vector with threshold for each bar, named with the variable in `barVar`

`barRange` (optional) range for the bars, either:

- a numeric vector of length 2, same range for all bars
- list with range for each bar, named with the variable in `barVar`

If not specified, the range of each `barVar` variable in `data` is used.

`filter` String with position of the filter boxes (`filter` parameter of the `datatable` function), 'top' by default. Set to 'none' to not included any filtering boxes.

`searchBox` Logical, if TRUE (FALSE by default) a general search box is included.

`pageLength` Numeric with number of records to include in one page, by default set to 10. Set to Inf to include all records.

`fixedColumns` List with fixed columns, see corresponding parameter in the `options` parameter of the `datatable` function.

`columnsWidth` Character vector with column width, of length 1 (used for all columns) or of length: `ncol(data)`

`options` List with additional `datatable` options. This parameter overwrites the default options set internally in the function (an indicative message mentions it if that is the case).

`expandVar` Character vector with expandable variables of `data`. These columns won't be included in the table, but displayed for each row when the '+' icon in the first column of the table will be clicked on.

`expandIdx` Matrix named with: 'row'/'column' containing row/column indices to expand.

`escape` Column(s) to escape in the table (e.g. containing raw HTML code), either character, numeric or logical of length 1. See corresponding parameter in the `datatable` function.

`rowGroup` This parameter is deprecated, please use `rowGroup` instead.

`rowGroupVar` Character vector with colname(s) of `data` containing variables to group rows by. This creates row header containing this column. Please note that the original row order in `data` is respected, so you might want to order rows based on the grouping variables upfront.

`vAlign` String with vertical alignment for the cells, 'top' by default.

`callback` String with custom Javascript callback function.

`buttons` DataTable buttons (passed to the 'buttons' element of the `options` parameter of `datatable`). See `getClinDTButtons` for the default options. To remove all buttons, set this parameter to NULL.

`scrollX` Logical, if TRUE (by default) a horizontal scrolling bar is included. Note: this differs from the `datatable` default (FALSE), because required for `bookdown::gitbook` output if table is too wide.

`file` (optional) String with name of html file to which the created DT should be exported.

`verbose` Logical, if TRUE (by default) informative messages are displayed, e.g. if specified `options` overwrite the internal default.

`...` Additional parameters for the `datatable` function, e.g table width.

Value

A datatable object.

Author(s)

Laure Cougnaud

Examples

```
data(dataADaMCDISCP01)
labelVars <- attr(dataADaMCDISCP01, "labelVars")

# example of simple adverse event table
dataAE <- dataADaMCDISCP01$ADAE
subjectsSafety <- subset(dataADaMCDISCP01$ADSL, SAFFL == "Y")$USUBJID

# compute counts of subjects presenting each AE
tableAE <- stats::aggregate(
USUBJID ~ AESOC:AEDECOD,
data = dataAE,
FUN = function(usubjid) length(unique(usubjid))
)
colnames(tableAE)[colnames(tableAE) == "USUBJID"] <- "N"
# and percentages
tableAE$perc <- round(tableAE$N/length(subjectsSafety)*100, 3)
# sort records in decreasing percentage
tableAE <- tableAE[order(tableAE$perc, decreasing = TRUE), ]

# extract new variables labels
tableAELabels <- getLabelVar(
var = colnames(tableAE),
labelVars = labelVars,
label = c(N = '# subjects', perc = "% subjects")
)
# 'colnames' for DT should be specified as c('new name' = 'old name', ...)
tableAELabelsDT <- setNames(names(tableAELabels), tableAELabels)

## create table with bar

# default:
getClinDT(
data = tableAE,
barVar = "perc",
colnames = tableAELabelsDT
)

# specify range for the bar
getClinDT(
data = tableAE,
filter = "none",
barVar = "perc",
barRange = c(0, 100),
```

```
colnames = tableAELabelsDT
)

# change color according to threshold
getClinDT(
data = tableAE,
filter = "none",
barVar = "perc",
barColorThr = seq(from = 0, to = 100, by = 25),
colnames = tableAELabelsDT
)

## group per system organ class (and decreasing N):
tableAESOC <- aggregate(N ~ AESOC, data = tableAE, FUN = sum)
tableAE$AESOC <- factor(tableAE$AESOC,
levels = tableAESOC[order(tableAESOC$N, decreasing = FALSE), "AESOC"]
)
tableAE <- tableAE[order(tableAE$AESOC, tableAE$perc, decreasing = TRUE), ]

getClinDT(
data = tableAE,
filter = "none",
barVar = "perc",
barRange = c(0, 100),
colnames = tableAELabelsDT,
rowGroupVar = "AESOC",
pageLength = Inf
)

# expand the subject ID column, will
# be accessible when clicking on the '+' button
# Format URL correctly with: 'escape',
# please note that indexing starts at 0!
getClinDT(
data = tableAE,
barVar = "perc",
colnames = tableAELabelsDT,
expandVar = "USUBJID",
escape = grep("USUBJID", colnames(tableAE))-1
)

# fix size for columns
getClinDT(
data = tableAE,
colnames = tableAELabelsDT,
fixedColumns = list(leftColumns = 1),
columnsWidth = c(0.1, 0.7, 0.1, 0.1),
width = "350px" # change dimension table
)

## Not run: # due to time constraint in CRAN

# change default buttons
```

```
getClinDT(
data = tableAE,
colnames = tableAELabelsDT,
# remove general filter
filter = "none",
# custom set of buttons
buttons = getClinDTButtons(type = c("csv", "excel", "pdf"))
)
# add button to select columns
getClinDT(
data = tableAE,
colnames = tableAELabelsDT,
# custom set of buttons
buttons = getClinDTButtons(typeExtra = "colvis")
)
# export pdf in landscape format
buttons <- getClinDTButtons(
opts = list(pdf = list(orientation = "landscape"))
)
getClinDT(
data = tableAE,
colnames = tableAELabelsDT,
# custom set of buttons
buttons = buttons
)

# hide the first column:
getClinDT(
data = tableAE,
nonVisibleVar = "AESOC"
)

# with specific caption
library(htmltools)
caption <- tags$caption(
"Number of subjects with adverse events grouped by system organ class.",
br(),
paste(
"Percentages are based on the total number of patients having",
"received a first study treatment."
)
)
getClinDT(
data = tableAE,
filter = "none",
barVar = "perc",
barRange = c(0, 100),
pageLength = Inf,
colnames = tableAELabelsDT,
rowGroupVar = "AESOC",
caption = caption
)
```

```
## End(Not run)
```

`getClinDTButtons` *Get a default set of buttons to be included in the interactive table for clinical data.*

Description

Get a default set of buttons to be included in the interactive table for clinical data.

Usage

```
getClinDTButtons(
  type = c("copy", "csv", "excel", "pdf", "print"),
  typeExtra = NULL,
  opts = NULL
)
```

Arguments

`type` Character vector with type of buttons, among:

- for export data:
  - 'copy' (by default): copy data to clipboard
  - 'csv' (by default): export selected data to a csv file
  - 'excel' (by default): export selected data to an Excel file
  - 'pdf' (by default): export data in a PDF file, in landscape format
  - 'print' (by default): extract the data with the print function of the browser

  For all these buttons, only the visible columns (selected by the show/hide button) are exported. The variables used for row grouping are always exported as well.
- to show/hide columns :
  - 'colvis': include a collection of buttons to show/hide specific columns. Specific columns that should not be listed should be defined in `nonVisibleVar` in `getClinDT`

`typeExtra` Character vector with type of button(s) that should be added to the default set of buttons.

`opts` List with extra opts for specific buttons. The list should be named with the button type.

Details

The 'colvis' button doesn't display the non visible columns.
These are defined internally with:

```
options = list(
  columnDefs = list(
    list(targets = [X], className = 'noVis')
  )
)
```

with [X] the index of the column(s) in Javascript notation (starting from 0)

Value

Nested list with default buttons to be passed on to 'buttons' option in the `getClinDT`.

Author(s)

Laure Cougnaud

`getColorPalette` *Get a color palette for clinical visualizations.*

Description

Get a color palette of specified length, either from a vector of names for the palette, or from a specified length.

Usage

```
getColorPalette(n = NULL, x = NULL, includeNA = FALSE, palette = clinColors)
```

Arguments

`n`	Integer of length 1, number of elements in palette.
`x`	Vector with elements used for palette. If factor, the levels are used, otherwise the unique elements of the vector. Missing values are automatically removed, excepted if `includeNA` is set to TRUE.
`includeNA`	Logical (FALSE by default), should NA elements be retained in the palette in case `x` is specified?
`palette`	A vector of custom colors, or a function returning this vector from a specific number of colors. Default is the the colorblind `viridis` color palette.

Value

Vector of colors, named with the elements in `x` if `x` is specified.

Author(s)

Laure Cougnaud and Michela Pasetto

Examples

```
# extract longest palette available
getColorPalette(n = 11)
# extract palette for a vector
getColorPalette(x = paste('treatment', 1:4))
# possibility to include missing values:
getColorPalette(x = c(NA_character_, "group1"), includeNA = FALSE)
getColorPalette(x = c(NA_character_, "group1"), includeNA = TRUE)
# change default settings
getColorPalette(n = 3, palette = c("red", "green", "grey"))
```

---

getLabelParamcd *Get label for a parameter code*

Description

This function gets the labele for a parameter code extracted from the 'PARAM' column.

Usage

```
getLabelParamcd(paramcd, data, paramcdVar = "PARAMCD", paramVar = "PARAM")
```

Arguments

paramcd	Character vector with parameter code(s).
data	Data.frame with data.
paramcdVar	String with column containing the paramcd parameter, 'PARAMCD' by default (for ADaM format).
paramVar	String with column containing the param parameter, 'PARAM' by default(for ADaM format).

Value

Named character vector with label for parameter code or paramcd if label is missing.

Author(s)

Laure Cougnaud

Examples

```
# for ADaM
data(dataADaMCDISCP01)
getLabelParamcd(paramcd = "CHOL", data = dataADaMCDISCP01$ADLBC)
# for SDTM
data(dataSDTMCDISCP01)
getLabelParamcd(
  paramcd = "ALB",
  data = dataSDTMCDISCP01$LB,
  paramcdVar = "LBTESTCD",
  paramVar = "LBTEST"
)
```

---

getLabelVar *Get label for a variable of the dataset*

Description

The label is extracted either (in this order):

1. if `label` is specified: from this label based on names, or directly from this label if `label` and `var` are of length 1 (if available)
2. if `labelVars` is specified: from the specified vector of labels, based on names (if available)
3. if `data` is specified: from the 'label' attribute of the corresponding column in `data` (if available)

If the label is not available, the input variable is returned.

Usage

```
getLabelVar(var, data = NULL, labelVars = NULL, label = NULL)
```

Arguments

`var`	Character vector with variables of interest.
`data`	Data.frame with data.
`labelVars`	Named character vector with variable labels (names are the variable code), usually extracted from `data`.
`label`	(Named) Character vector with user-specified label for `var`. Label is extracted based on names if variable is available. If `var` is of length 1, `label` can also be specified as an unnamed character.

Value

Named character vector with label, `var` is no label is available

Author(s)

Laure Cougnaud

Examples

```
data(dataADaMCDISCP01)
labelVars <- attr(dataADaMCDISCP01, "labelVars")

# (upon reading the data with haven: attributes should directly available in each column)
getLabelVar(data = dataADaMCDISCP01, var = "AEREL")

# but if the data as data.frame is subsetted, label is lost
# so better to use 'labelVars':
getLabelVar(var = "AEREL", labelVars = labelVars)
```

---

getLabelVars *Get label of the variables in SAS dataset(s)*

Description

Get label of the variables in SAS dataset(s)

Usage

```
getLabelVars(data, labelVars = NULL)
```

Arguments

data	Data.frame with SAS dataset(s) or list of those.
labelVars	(optional) Named character vector with additional labels.

Value

Named vector with variable labels.

Author(s)

Laure Cougnaud

Examples

```
data(dataADaMCDISCP01)
labelVars <- attr(dataADaMCDISCP01, "labelVars")

# extract label for all variables from specified datasets:
getLabelVars(data = dataADaMCDISCP01[c("ADLBC", "ADVS")], labelVars = labelVars)

# extracted from specified labelVars, e.g. to specify custom label for specific variable(s)
```

```
labelVarsCustom <- getLabelVars(
  data = dataADaMCDISCP01,
  labelVars = c(USUBJID = "Subject identifier for my study")
)
labelVarsCustom["USUBJID"]
```

`getLinetypePalette` *Get a linetype palette for clinical visualizations.*

Description

Get a linetype palette of specified length, either from a vector of names for the palette, or from a specified length.

Usage

```
getLinetypePalette(
  n = NULL,
  x = NULL,
  includeNA = FALSE,
  palette = clinLinetypes
)
```

Arguments

`n`	Integer of length 1, number of elements in palette.
`x`	Vector with elements used for palette. If factor, the levels are used, otherwise the unique elements of the vector. Missing values are automatically removed, excepted if `includeNA` is set to TRUE.
`includeNA`	Logical (FALSE by default), should NA elements be retained in the palette in case `x` is specified?
`palette`	A vector of custom linetypes, or a function returning this vector from a specific number of linetypes. Default is the `clinLinetypes` linetype palette.

Details

Note that 7 unique symbols are available at maximum (replicated if necessary).

Value

Vector with linetypes, named with the elements in `x` if `x` is specified.

Author(s)

Laure Cougnaud and Michela Pasetto

Examples

```
# extract longest linetype palette available
getLinetypePalette(n = 6)
# extract palette for a vector
getLinetypePalette(x = paste('treatment', 1:4))
# include missing
getLinetypePalette(x = c(NA_character_, "group1"), includeNA = TRUE)
getLinetypePalette(x = c(NA_character_, "group1"), includeNA = FALSE)
# set custom linetypes
lty <- getColorPalette(n = 3, palette = c("twodash", "dashed"))
```

`getPaletteCDISC` *Get standard palette for typical CDISC variables.*

Description

The extraction of the palette elements is case-insensitive.

Usage

```
getPaletteCDISC(x, var, type, palette = NULL)
```

Arguments

`x`	Character vector of factor with variable to consider. The palette is built based on the unique elements of this vector, or levels if `x` is a factor.
`var`	String with type of variable, among: • 'NRIND': Normal Reference Range Indicator
`type`	String with type of palette: • 'shape': shape/symbol palette • 'color': color palette
`palette`	(optional) Named vector with extra palette, e.g. to specify elements for non-standard categories. This palette is combined with the standard palette.

Details

The order of the palette depends on the type of the input variable (`x`):

- if a factor is specified, the palette is ordered based on its levels
- if a character vector is specified, the elements from the internal standard palette are used first, the remaining elements are then sorted alphabetically.

Value

Named vector with palette.

Author(s)

Laure Cougnaud

Examples

```
## palette for reference range indicator variables

xRIND <- c("LOW", "HIGH", "NORMAL", "NORMAL", "NORMAL", "ABNORMAL")

# get standard palette
getPaletteCDISC(x = xRIND, var = "NRIND", type = "shape")
getPaletteCDISC(x = xRIND, var = "NRIND", type = "color")

# in case extra categories are specified:
xRIND <- c(xRIND, "High Panic")
# the symbols are set to numeric symbols
getPaletteCDISC(xRIND, var = "NRIND", type = "shape")
# use shapePalette to specify symbols for extra categories
getPaletteCDISC(xRIND, var = "NRIND", type = "shape", palette = c("High Panic" = "\u2666"))

# palette is case-insensitive
xRIND <- c("Low", "High", "Normal", "Normal", "Normal")
getPaletteCDISC(xRIND, var = "NRIND", type = "shape")
```

---

getSetDiff *Get additions/removals*

Description

Get only additions and removals from two data sets (`data.table` objects). The additions/removals are extracted as `x` vs `y`. This function assumes that the objects `x` and `y` don't share identical rows.

Usage

```
getSetDiff(x, y, referenceVars)
```

Arguments

`x`	A `data.table` object
`y`	A `data.table` object
`referenceVars`	character vector of the columns in the data that are the used as reference for the comparison. If not specified, all columns present both in `newData` and `oldData` are considered.

Value

A `data.table` object with the additions/removals with respect of the comparison between `x` vs `y`.

`getShapePalette` *Get a shape palette for clinical visualizations.*

Description

Get a shape palette of specified length, either from a vector of names for the palette, or from a specified length.

Usage

```
getShapePalette(
  n = NULL,
  x = NULL,
  includeNA = FALSE,
  asText = FALSE,
  palette = if (asText) {
     clinShapesText
 } else {
     clinShapes
 }
)
```

Arguments

`n`	Integer of length 1, number of elements in palette.
`x`	Vector with elements used for palette. If factor, the levels are used, otherwise the unique elements of the vector. Missing values are automatically removed, excepted if `includeNA` is set to TRUE.
`includeNA`	Logical (FALSE by default), should NA elements be retained in the palette in case `x` is specified?
`asText`	Logical (FALSE by default), should the palette be expressed as integer (base R plot and ggplot2 compatible) or in text format (e.g. required if combined with unicode symbols in ggplot2)?
`palette`	A vector of custom shapes, or a function returning this vector from a specific number of shapes. The vector should be a character if `asText` is set to TRUE. Default is the `clinShapes` shape palette, or `clinShapesText` if `asText` is set to TRUE.

Details

Note that 19 unique symbols are available at maximum (replicated if necessary).

Value

Vector of shapes, named with the elements in `x` if `x` is specified.

Author(s)

Laure Cougnaud and Michela Pasetto

Examples

```
#' extract longest shape palette available
getShapePalette(n = 19)
# extract palette for a vector
getShapePalette(x = paste('treatment', 1:4))
# include missing
getShapePalette(x = c(NA_character_, "group1"), includeNA = TRUE)
getShapePalette(x = c(NA_character_, "group1"), includeNA = FALSE)
# change default settings
getShapePalette(x = paste('treatment', 1:3), palette = c("circle", "triangle"))
# get symbols as 'text' (e.g. to be combined with Unicode in ggplot2)
getShapePalette(x = paste('treatment', 1:4), asText = TRUE)
```

---

knitPrintListObjects *Include a list of objects in a knitr document*

Description

Each object is included (internally) in a separated chunk, so different chunk options can be set for each object.

Usage

```
knitPrintListObjects(
  xList,
  generalLabel = "objectsList",
  labels = paste0(generalLabel, seq_along(xList)),
  titles = NULL,
  titleLevel = 2,
  printObject = FALSE,
  ...
)
```

Arguments

xList	List of objects to print.
generalLabel	String with general label for the chunks, used to build the labels. The labels are constructed as 'generalLabel[i]', with i the list index. Only use if labels is not specified.
labels	Character vector with labels, one for each chunk. This is also used to define file names for plots exported in the document (e.g. via opts_chunk$set(dev = "png")).
titles	Character vector with section titles, one for each chunk.

titleLevel	Integer with level for section header, 1 for top-level section header.
printObject	Logical, if TRUE (FALSE by default), each object within xList is explicitely printed with the print function.
...	any knitr chunk parameters (excepted 'results', set to 'asis' and 'echo' set to FALSE internally). See knitr[opts_chunk] for further details on available options. Each parameter can be specified for each element in the list separately: by specifying a vector with the same length than the list; or for all elements at once: by specifying a vector of length 1 (in this case it will be replicated).

Details

This function should be called within a chunk with the following option: results = 'asis'.

Value

No returned value, a text is printed with chunk content

Author(s)

Laure Cougnaud

Examples

```
## Not run:

# Note: the following code should be included
# within a chunk of a knitr (e.g. RMarkdown) document
# to include a list of objects in the Rmarkdown output

# list of flextable objects
library(flextable)
listTables <- list(flextable(iris), flextable(cars))
knitPrintListObjects(
xList = listTables,
titles = c("Iris dataset", "Cars dataset")
)


## End(Not run)
```

---

knitPrintListPlots *Include a list of plots in a knitr document*

Description

Each plot is included (internally) in a separated chunk, so different chunk options can be set for each plot.
For example, plots can be created with different figure height or width (see examples).

Usage

```
knitPrintListPlots(
  plotsList,
  generalLabel = "plotsList",
  type = c("ggplot2", "plotly"),
  ...
)
```

Arguments

plotsList	list of plots, e.g. ggplot objects from the ggplot2 package or from the plotly packages.
generalLabel	general label for the chunks, used to build the labels. The labels are constructed as 'generalLabel[i]', with i the plot number (from sequence spanning the length of plotsList). Only use if labels is not specified.
type	string with plot type: 'ggplot2' or 'plotly'
...	Arguments passed on to knitPrintListObjects labels Character vector with labels, one for each chunk. This is also used to define file names for plots exported in the document (e.g. via opts_chunk$set(dev = "png")). titles Character vector with section titles, one for each chunk. titleLevel Integer with level for section header, 1 for top-level section header.

Details

This function should be called within a chunk with the following option: results = 'asis'.
Note that a (one-level) list of plotly plots can also be included directly via htmltools::tagList(listPlots), but without the possibility to have different chunk option for each plot.

Value

No returned value, a text is printed with chunk content

Author(s)

Laure Cougnaud

Examples

```
## Not run:

# Note: the following code should be included
# within a chunk of a knitr (e.g. RMarkdown) document
# to include a list of figures in the Rmarkdown output
data(iris)

## Static plots
```

```
library(ggplot2)
plotsListStatic <- list(
point = ggplot(data = cars, aes(x = speed, y = dist)) + geom_point(),
line = ggplot(data = cars, aes(x = speed, y = dist)) + geom_line()
)
# with general label (used to name exported figure)
knitPrintListPlots(
plotsList = plotsListStatic,
generalLabel = "scatter-cars"
)
# with label for each plot (used to name exported figure)
knitPrintListPlots(
plotsList = plotsListStatic,
labels = names(plotsListStatic)
)
# with section header (header of level 1 in Markdown)
knitPrintListPlots(
plotsList = plotsListStatic,
titles = names(plotsListStatic),
titleLevel = 3
)
# with caption for each figure
knitPrintListPlots(
plotsList = plotsListStatic,
fig.cap = names(plotsListStatic)
)

# specify dimension for each figure
knitPrintListPlots(
plotsList = plotsListStatic,
# first plot has width of 3, second of 6
fig.width = c(3, 6),
# both plots have a height of 6
fig.height = 6
)

## Interactive plots

library(plotly)
plotsListInteractive <- list(
point = plot_ly(data = cars, x = ~speed, y = ~dist, type = "scatter", mode = "marker"),
line = plot_ly(data = cars, x = ~speed, y = ~dist, type = "scatter", mode = "line")
)

# with titles
knitPrintListPlots(
plotsList = plotsListInteractive,
type = "plotly",
titles = names(plotsListInteractive),
titleLevel = 3
)
```

```
## End(Not run)
```

`loadDataADaMSDTM` *Load data from ADaM/SDTM file(s).*

Description

Load data set from SAS format into R data.frames.

Usage

```
loadDataADaMSDTM(
  files,
  convertToDate = FALSE,
  dateVars = "DTC$",
  verbose = TRUE,
  encoding = "UTF-8",
  ...
)
```

Arguments

`files`	Character vector with path to ADaM or SDTM file(s). Currently only import of files with extension: 'sas7bdat' or 'xpt' are supported.
`convertToDate`	logical, if TRUE columns with date/time are converted to `POSIXct` format, which stores calendar date/time in R. Please note that most of the time this is not necessary, as date variables are automatically imported via the `haven` package if encoded correctly in the dataset.
`dateVars`	vector of columns in `data` containing date/time, or pattern for this columns. By default all columns ending with 'DTC' are used (dateVars is: 'DTC$').
`verbose`	logical, if TRUE (by default) progress messages are printed during execution.
`encoding`	String with encoding, only used if `files` is of extension: 'sas7bdat', 'UTF-8' by default.
`...`	Additional parameters for the `read_sas` or `read_xpt` functions, depending on the input file type.

Details

While creating the R data.frames, if date/time variables are present, those are converted into to R date/time class (see `convertToDateTime`) function.

The labels of the ADaM/SDTM data sets are attached as attributes of the R data.frame.

Value

List of data.frame with data of each ADAM file (if not empty), with special attributes 'labelVars': named vector with label of the variables. Each data.frame contains an additional column called 'dataset' specifying the name of the `files` it was read from.

Author(s)

Laure Cougnaud

Examples

```
## Not run:
dataFromSAS7bdat <- loadDataADaMSDTM(files = "ae.sas7bdat")
attr(dataFromSAS7bdat, "labelVars") # column labels
dataFromXpt <- loadDataADaMSDTM(files = c("ae.xpt", "dm.xpt"))
attr(dataFromXpt, "labelVars") # column labels

## End(Not run)
```

---

mergeDiffWithData *Merge the 'diff.data' object from* compareDiff *with the original* newData *or* oldData.

Description

The newData/oldData are merged with diffData based on the columns of diffData excepted 'Comparison type' and 'Version'.

Usage

```
mergeDiffWithData(diffData, newData, oldData)
```

Arguments

diffData	Object of class 'diff.data' containing differences between datasets, as returned by the compareDiff function.
newData	data.frame object representing the new data
oldData	data.frame object representing the old data

Value

The newData or oldData (as a data frame object) with the extra column 'Comparison type' specifying the type of change, either:

- 'Change': record present in both dataset based on the reference variables, but with changes in the changeable variables
- 'Addition': records present in new but not in old data
- 'Removal': records present in old but not in new data
- 'Identical': records identical in the old and new datasets (on both the reference and changeable variables)

mergeInputDiff *Custom merge of difference data with input data*

Description

Custom merge (left join) of difference data with some input data

Usage

```
mergeInputDiff(diffData, inputData, typeData, colsBy)
```

Arguments

diffData	A data.table object as output from compareDiff.
inputData	A data.table object. For instance, the newData or the oldData argument from compareTables.
typeData	String with type of data, as "new" for newData or "old" for oldData.
colsBy	Character vector of columns for doing the merge by.

Value

A data.table object. The inputData is joined with the columns Comparison type and Version from the diffData argument.

reorderColumns *Function for reordering columns*

Description

Function for reordering columns

Usage

```
reorderColumns(data, vars)
```

Arguments

data	A data.frame
vars	Named vector indicating the position in the data frame of the specified variable

Value

The same data.frame specified in data, with ordered columns.

Examples

```
someData <- data.frame(
"Col1" = c(1, 2),
"Col2" = c(2, 3),
"Col3" = c(3, 4)
)
reorderColumns(
data = someData,
vars = c("Col3" = 1)
)
```

`roundHalfUp` *Round a number with 'rounding up' strategy for rounding off a 5*

Description

This function rounds a number for a specified number of digits. It rounds off to the highest number for a 5. The default R `round` function rounds to the 'even digit' in case of rounding off a 5 (see 'Details' section in ? `round`). This function instead rounds up to the nearest number for a 5. It mimics a similar rounding strategy used in SAS. See examples for the difference between `round` and 'roundHalfUp' below.

Usage

```
roundHalfUp(x, digits = 0)
```

Arguments

`x`	Numeric vector to round.
`digits`	Integer with number of digits to consider, 0 by default.

Value

Rounded numeric vector.

Author(s)

stackoverflow question 6461209

Examples

```
# numbers are rounded to the closest even number in case of .5
# with the round 'base' function
round(0.45, 1)
# 'roundHalfUp' always round to the next highest number in case of .5
roundHalfUp(0.45, 1)
# rounding is the same for uneven number:
round(0.55, 1)
```

```
roundHalfUp(0.55)
# other examples
round(1.456e-2, digits = 3)
round(1.456e-2, digits = 2)
round(1.456e-2, digits = 1)
```

`roundHalfUpTextFormat` *Round a number with 'round-up' strategy for rounding off a 5 in text format*

Description

This function rounds numbers with a 'round-up' strategy for rounding off a 5. The function rounds for a specified number of digits and format number to a: 'xxx.xxx' text.

Usage

```
roundHalfUpTextFormat(x, digits = 0)
```

Arguments

`x`	Numeric vector to round.
`digits`	Integer with number of digits to consider, 0 by default.

Details

The following workflow is used:

1. numbers are rounded with the `roundHalfUp` function, see the `? roundHalfUp` for more details on the rounding strategy
2. round numbers are formatted to character in the format: 'xxx.xxx' with pads leading zeros

Value

A character vector with the rounded number. NA values are returned as 'NA' as string.

Author(s)

Laure Cougnaud and Michela Pasetto

See Also

roundHalfUp for the rounding customization.

Examples

```
# number of digits higher than number of decimal
roundHalfUpTextFormat(x = c(0.345, 0.567, -0.98), digits = 2)
# number of digits lower than number of decimal
roundHalfUpTextFormat(x = c(0.345, 0.567, -0.98), digits = 0)
# by default, 'digits' is 0!
roundHalfUpTextFormat(x = c(0.345, 0.567, -0.98))
# padding zeros
roundHalfUpTextFormat(1.23, 10)
```

---

shapePaletteNRIND *Shape palette for a standard CDISC Normal/Reference Range Indicator.*

Description

These symbols should be supported in Windows and Linux.

Usage

```
shapePaletteNRIND
```

Format

A named character vector with shape symbol for typical Normal Reference Range Indicator variable:

- "LOW": filled down-pointing arrow (25)
- "NORMAL": filled circle (21)
- "HIGH": filled up-pointing arrow (24)
- "ABNORMAL": diamond (18)
- "UNKNOWN" or 'NA': cross (3)
- "NA": cross (3)

---

simpleCap *Capitalize the first letter of a word/sentence.*

Description

This implementation is inspired from the help of the `toupper` function.

Usage

```
simpleCap(x, onlyFirst = TRUE, rev = FALSE)
```

Arguments

x	Character vector to capitalize
onlyFirst	Logical, if TRUE (by default) capitalize the first letter of the first word only. Otherwise, capitalize the first letters of all words of the sentence. See also link[tools]{toTitleCase} for a more syntax-friendly implementation.
rev	Logical, if TRUE (FALSE by default), set first letter to lower case (otherwise upper case)

Value

Character vector with first letter capitalized

Author(s)

author of the 'toupper' function?

See Also

link[tools]{toTitleCase}

Examples

```
# capitalize only the first word of the sentence
simpleCap(x = "this is the caption of my figure.")
# capitalize all words
simpleCap(x = "this is the caption of my figure.", onlyFirst = FALSE)
# opposite: set the first letter of the first word to lower case
simpleCap(x = "This is the caption of my figure.", rev = TRUE)
```

Index