Package ‘clinDataReview’

June 18, 2024

Type Package

Title Clinical Data Review Tool

Version 1.6.1

Date 2024-06-17

Description Creation of interactive tables, listings and figures ('TLFs')
and associated report for exploratory analysis of data in a clinical trial,
e.g. for clinical oversight activities.
Interactive figures include sunburst, treemap, scatterplot, line plot and
barplot of counts data.
Interactive tables include table of summary statistics
(as counts of adverse events, enrollment table) and listings.
Possibility to compare data (summary table or listing) across two data batches/sets.
A clinical data review report is created via study-specific configuration
files and template 'R Markdown' reports contained in the package.

Imports bookdown, clinUtils (>= 0.1.0), crosstalk, data.table,
ggplot2, grid, haven, htmltools, htmlwidgets, knitr, jsonlite,
jsonvalidate, methods, plotly, plyr, rmarkdown, stats, stringr,
utils, tools, yaml, xml2, xfun, base64enc

Suggests countrycode, inTextSummaryTable (>= 3.1.0),
patientProfilesVis (>= 0.12.0), testthat, DT, scales, grDevices

SystemRequirements pandoc (to create a clinical data review report)

URL `https://github.com/openanalytics/clinDataReview`

BugReports `https://github.com/openanalytics/clinDataReview/issues`

License MIT + file LICENSE

VignetteBuilder knitr

RoxygenNote 7.3.1

NeedsCompilation no

Author Laure Cougnaud [aut, cre],
Michela Pasetto [aut],
Lennart Tuijnder [aut],
Adriaan Blommaert [aut],

Arne De Roeck [ctb, rev] (rev: tests),
Open Analytics [cph]

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

Repository CRAN

Date/Publication 2024-06-18 09:10:05 UTC

Contents

`addDateOfReportRun` *Add date of report running*

Description

Add the today’s date of when the report runs to the info of the metadata.

Usage

```
addDateOfReportRun(summaryInfo)
```

Arguments

`summaryInfo`	matrix, see output from `getMetadata`.

Value

A matrix, same as input `summaryInfo` with an extra row with the date of today.

`addFacetPanel` *Add facet-panel to single plotly plot.*

Description

Add facet-panel to single plotly plot.

Usage

```
addFacetPanel(
  pl,
  panelLab,
  panelWidth = 20,
  fontSize = 15,
  side = c("top", "right")
)
```

Arguments

`pl`	`plotly` object.
`panelLab`	text to be shown in the facet panel
`panelWidth`	thickness of the panel in pixels.
`fontSize`	fontsize of `facetText`
`side`	the side of the plot to show the panel (currently only right panels are implemented.)

Details

plot title clipping.

In case side = 'top', the plot title (eg. layout(title = "title")) will clip with the top panel.

Resolve this with the following configurations: (once all the subplots have already been combined) `layout( # place the title at absolute top of the page title = list(text = "title", yref = "container", y = 1)) # If font size = 15 roughly equal to 20 pixels. margin = list(t = panelWidth + heightTitleTextInPixels)`

Value

plotly object with the facet panel added.

Author(s)

lennart tuijnder

`addLayerToScatterPlot` *Helper function to add layer to scatter plot*

Description

Helper function to add layer to scatter plot

Usage

```
addLayerToScatterPlot(
  gg,
  aesVar,
  pars,
  generalPars,
  layerFunction,
  useHandlers = FALSE
)
```

Arguments

`gg`	`ggplot` object
`aesVar`	layers specific aesthetics list of layer specific aesthetics
`pars`	list of parameters specific to the layer `aes`
`generalPars`	overall, not layer specific parameters can be overwritten by `pars`
`layerFunction`	function to use for adding the layer e.g. `geom_line`
`useHandlers`	if `TRUE` we use handlers to repress the expected warning: `Hover is set via the 'text' aesthetic in ggplot`, we need to pass this aesthetic to have it available in plotly even though it is not used by geom_point. Defaults to `FALSE`

Value

`ggplot` object

Author(s)

Adriaan Blommaert Laure Cougnaud

`addReferenceLinesClinDataPlot`

Add reference (horizontal/vertical/diagonal) lines to a clinical data plot

---

Description

Add reference (horizontal/vertical/diagonal) lines to a clinical data plot

Usage

```
addReferenceLinesClinDataPlot(
  gg,
  data,
  xVar,
  yVar,
```

```
    xLim = NULL,
    yLim = NULL,
    refLinePars = NULL,
    facetPars = NULL
  )
```

Arguments

`gg`	`ggplot` object.
`data`	Data.frame with data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLim`, `yLim`	Numeric vector of length 2 with limits for the x/y axes.
`refLinePars`	(optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains: • aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`. • 'label': (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.
`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.

Value

Updated `ggplot` object.

Author(s)

Laure Cougnaud

`addSelectBtn` *Add selection box(es) to a plotly plot.*

Description

Add selection box(es) to a plotly plot.

Usage

```
addSelectBtn(
  data,
  pl,
  selectVars = NULL,
  selectLab = NULL,
  labelVars = NULL,
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  keyVar
)
```

Arguments

`data`	`SharedData` object used for the plot.
`pl`	`plotly` object.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`labelVars`	Named character vector containing variable labels.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`keyVar`	String with unique key variable, identifying unique group for which the link between the table and the plot should be done.

Value

if `selectVars` is specified: a `browsable` object combining the select buttons and the `plotly` object. Otherwise, the input `plotly` object.

`addWatermark` *Add a watermark to a* `plotly` *object.*

Description

Add a watermark to a `plotly` object.

Usage

```
addWatermark(pl, watermark = NULL)
```

Arguments

`pl`	`plotly` object.
`watermark`	(optional) String with path to a file containing a watermark.

Value

`plotly` object with a watermark (if specified)

See Also

Other watermark helpers: `getWatermark()`

Examples

```
watermark <- tempfile(pattern = "watermark", fileext = ".png")
getWatermark(file = watermark)
addWatermark(pl = plotly::plot_ly(), watermark = watermark)
```

`annotateData` *Annotate a dataset.*

Description

Standard annotation variables are available via the parameter `annotType`. Custom dataset/variables of interest are specified via the `annotDataset`/`annotVar` parameters.

Usage

```
annotateData(
  data,
  dataPath = ".",
  annotations,
  subjectVar = "USUBJID",
  verbose = FALSE,
  labelVars = NULL,
  labelData = "data"
)
```

Arguments

`data`	Data.frame with input data to annotate.
`dataPath`	String with path to the data.
`annotations`	Annotations (or list of those) either as a: • string with standard annotation type, among: – demographics: standard variables from the demographics data (DM or ADSL) are extracted

  - exposed_subjects: a logical variable: `EXFL` is added to `data`, identifying exposed subjects, i.e. subjects included in the exposure dataset (EX/ADEX) dataset and with non empty and non missing start date ('EXSTDTC', 'STDY' or 'ASTDY')
  - functional_groups_lab: a character variable: 'LBFCTGRP' is added to `data` based on standard naming of the parameter code ('PARAMCD' or 'LBTESTCD' variable)
- list of custom annotation, with:
  - (optional) annotation dataset, either:
    - 'dataset': String with name of the annotation dataset, e.g. 'ex' to import data from the file: '[dataset].sas7bdat'in `dataPath`
    - 'data': Data.frame with annotation dataset

    The input `data` is used if 'data' and 'dataset' are not specified.
  - 'vars': Either:
    - Character vector with variables of interest from annotation dataset. If not specified, all variables of the dataset are considered.
    - String with new variable name computed from `varFct`
  - 'varFct': (optional) Either:
    - function of `data` or string containing such function (e.g. 'function(data) ...')
    - string containing manipulations from column names of `data` (e.g. 'col1 + col2')

    used to create a new variable specified in `vars`.
  - 'filters': (optional) Filters for the **annotation dataset**, see `filters` parameter of `filterData`.
    The annotation dataset is first filtered, before being combined to the input `data`, such as only the records retained in the annotation dataset will be annotated in the output `data`. Other records will have missing values in the annotated variables.
  - 'varLabel': (optional) label for new variable in case `varFct` is specified.
  - 'varsBy': (optional) Character vector with variables used to merge input data and the annotation dataset. If not specified:
    - if an external dataset (`dataset/data`) is specified: `subjectVar` is used
    - otherwise: annotation dataset and input data are merged by rows IDs

`subjectVar`	String with subject ID variable, 'USUBJID' by default.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.
`labelVars`	Named character vector containing variable labels of `data`. This will be updated with the labels of the extra annotation variables (in `attr(output, 'labelVars')`).
`labelData`	(optional) String with label for input `data`, that will be included in progress messages.

Value

Annotated `data`. If `labelVars` is specified, the output contains an extra attribute: 'labelVars' containing updated `labelVars` (accessible via: in `attr(output, 'labelVars')`).

Examples

```
library(clinUtils)

data(dataADaMCDISCP01)

dataLB <- dataADaMCDISCP01$ADLBC
dataDM <- dataADaMCDISCP01$ADSL
dataAE <- dataADaMCDISCP01$ADAE

labelVars <- attr(dataADaMCDISCP01, "labelVars")

# standard annotations:
# path to dataset should be specified via: 'pathData'
## Not run:
annotateData(dataLB, annotations = "demographics", pathData = ...)

## End(Not run)

# add all variables in annotation data (if not already available)
head(annotateData(dataLB, annotations = list(data = dataDM)), 1)

# only variables of interest
head(annotateData(dataLB, annotations = list(data = dataDM, vars = c("ARM", "ETHNIC"))), 1)

# filter annotation dataset
dataAnnotated <- annotateData(dataLB,
annotations = list(
data = dataDM,
vars = c("ARM", "ETHNIC"),
filters = list(var = "ARM", value = "Placebo")
)
)
head(subset(dataAnnotated, ARM == "Placebo"), 1)
head(subset(dataAnnotated, is.na(ARM)), 1)

# worst-case scenario: add a new variable based on filtering condition
dataAE$AESEV <- factor(dataAE$AESEV, levels = c('MILD', "MODERATE", "SEVERE"))
dataAEWC <- annotateData(
data = dataAE,
annotations = list(
vars = "WORSTINT",
# create new variable: 'WORSTINT'
# with TRUE if maximum toxicity grade per subject/test
# (if multiple, they are all retained)
filters = list(
var = "AESEV",
# max will take latest level in a factor
```

```
# (so 'MODERATE' if 'MILD'/'MODERATE' are available)
valueFct = function(x) x[which.max(as.numeric(x))],
varsBy = c("USUBJID", "AEDECOD"),
keepNA = FALSE,
varNew = "WORSTINT",
labelNew = "worst-case"
)
),
labelVars = labelVars,
verbose = TRUE
)
attr(dataAEWC, "labelVars")["WORSTINT"]

# add a new variable based on a combination of variables:
dataLB <- annotateData(dataLB,
annotations = list(vars = "HILORATIO", varFct = "A1HI / A1LO")
)

# add a new variable based on extraction of a existing variable
# Note: slash should be doubled when the function is specified as text
dataLB <- annotateData(dataLB,
annotations = list(vars = "PERIOD", varFct = "sub('.* Week (.+)', 'Week \\\\1', AVISIT)")
)

# multiple annotations:
dataAnnotated <- annotateData(dataLB,
annotations = list(
list(data = dataDM, vars = c("ARM", "ETHNIC")),
list(data = dataAE, vars = c("AESEV"))
)
)
head(dataAnnotated, 1)
```

`barplotClinData` *Barplot visualization of clinical data.*

Description

Barplot visualization of clinical data.

Usage

```
barplotClinData(
  data,
  xVar,
  yVar,
  xLab = getLabelVar(xVar, labelVars = labelVars),
  xLabVar = NULL,
  yLab = getLabelVar(yVar, labelVars = labelVars),
  yLabVar = NULL,
```

```
    colorVar = NULL,
    colorLab = getLabelVar(colorVar, labelVars = labelVars),
    colorPalette = NULL,
    barmode = "group",
    title = paste(c(paste(yLab, "vs", xLab), titleExtra), collapse = "<br>"),
    titleExtra = NULL,
    caption = NULL,
    subtitle = NULL,
    labelVars = NULL,
    width = NULL,
    height = NULL,
    hoverVars,
    hoverLab,
    textVar = NULL,
    pathVar = NULL,
    pathLab = getLabelVar(pathVar, labelVars = labelVars),
    table = FALSE,
    tableVars,
    tableLab,
    tableButton = TRUE,
    tablePars = list(),
    id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
    selectVars = NULL,
    selectLab = getLabelVar(selectVars, labelVars = labelVars),
    watermark = NULL,
    verbose = FALSE
  )
```

Arguments

`data`	Data.frame with data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLab`	String with label for `xVar`.
`xLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the x-axis.
`yLab`	String with label for `xVar`.
`yLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the y-axis.
`colorVar`	(optional) String with color variable.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`barmode`	String with type of barplot, either: 'group' or 'stack' (see parameter in `layout`).
`title`	String with title for the plot.

`titleExtra`	String with extra title for the plot (appended after `title`).
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`labelVars`	Named character vector containing variable labels.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.
`textVar`	(optional) String with a text variable, that will be displayed outside of each bar.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be: • compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`watermark`	(optional) String with path to a file containing a watermark.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other visualizations of summary statistics for clinical data: `boxplotClinData()`, `errorbarClinData()`, `plotCountClinData()`, `sunburstClinData()`, `treemapClinData()`

Examples

```
library(clinUtils)

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

dataAE <- dataADaMCDISCP01$ADAE
dataDM <- dataADaMCDISCP01$ADSL

## example of basic barplot:

# treemap takes as input table with counts

if (requireNamespace("inTextSummaryTable", quietly = TRUE)) {

# total counts: Safety Analysis Set (patients with start date for the first treatment)
dataTotal <- subset(dataDM, RFSTDTC != "")

# compute adverse event table

tableAE <- inTextSummaryTable::computeSummaryStatisticsTable(
data = dataAE,
rowVar = c("AEBODSYS", "AEDECOD"),
rowOrder = "total",
dataTotal = dataTotal,
labelVars = labelVars,
stats = inTextSummaryTable::getStats("count")
)

dataPlot <- subset(tableAE, AEDECOD != "Total")

dataPlot$n <- as.numeric(dataPlot$n)

# create plot
```

```
barplotClinData(
data = dataPlot,
xVar = "AEDECOD",
yVar = "n", yLab = "Number of patients with adverse events",
labelVars = labelVars
)

# add number on top of the bars
barplotClinData(
data = dataPlot,
xVar = "AEDECOD",
yVar = "n", yLab = "Number of patients with adverse events",
textVar = "n",
labelVars = labelVars
)

# add a selection box
if(interactive()){
  barplotClinData(
    data = dataPlot,
    xVar = "AEDECOD",
    yVar = "n", yLab = "Number of patients with adverse events",
    labelVars = labelVars,
    selectVars = "AEBODSYS"
  )
}

## Not run:

# display percentage of events per severity
tableAEBySeverity <- inTextSummaryTable::computeSummaryStatisticsTable(
data = dataAE,
rowVar = c("AEDECOD", "AESEV"),
dataTotal = dataTotal,
labelVars = labelVars,
statsPerc = "statm",
stats = inTextSummaryTable::getStats("%m"),
dataTotalPerc = dataAE,
rowVarTotalPerc = "AEDECOD"
)
barplotClinData(
data = tableAEBySeverity,
xVar = "AEDECOD", xLab = "Adverse event term",
yVar = "statPercm", yLab = "Percentage of adverse events",
labelVars = labelVars,
colorVar = "AESEV", barmode = "stack",
hoverVar = c("AEDECOD", "AESEV", "statN", "statm", "statPercm"),
hoverLab = c(
labelVars["AEDECOD"],
labelVars["AESEV"],
statN = "Number of patients",
statm = "Number of events",
statPercm = "Percentage of events"
```

```
),
textVar = "%m",
# add subtitle
subtitle = "Group: severity"
)


## End(Not run)

}
```

`boxplotClinData` *Boxplot interactive plot.*

Description

Boxplot interactive plot.

Usage

```
boxplotClinData(
  data,
  xVar,
  yVar,
  xLab = getLabelVar(xVar, labelVars = labelVars),
  xLabVar = NULL,
  yLab = getLabelVar(yVar, labelVars = labelVars),
  yLabVar = NULL,
  colorVar = NULL,
  colorLab = getLabelVar(colorVar, labelVars = labelVars),
  colorPalette = NULL,
  facetVar = NULL,
  facetLab = getLabelVar(facetVar, labelVars = labelVars),
  ncol = 1L,
  title = paste(c(paste(yLab, "vs", xLab), titleExtra), collapse = "<br>"),
  titleExtra = NULL,
  subtitle = NULL,
  caption = NULL,
  labelVars = NULL,
  width = NULL,
  height = NULL,
  hoverVars,
  hoverLab,
  pathVar = NULL,
  pathLab = getLabelVar(pathVar, labelVars = labelVars),
  idVar = "USUBJID",
  idLab = getLabelVar(idVar, labelVars = labelVars),
  table = FALSE,
```

```
  tableVars,
  tableLab,
  tableButton = TRUE,
  tablePars = list(),
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  watermark = NULL,
  verbose = FALSE
)
```

Arguments

`data`	Data.frame with data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLab`	String with label for `xVar`.
`xLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the x-axis.
`yLab`	String with label for `xVar`.
`yLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the y-axis.
`colorVar`	(optional) String with color variable.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`facetVar`	(optional) String with facet variable.
`facetLab`	String with label for `facetVar`.
`ncol`	single-length integer denoting the number of columns for the facetting.
`title`	String with title for the plot.
`titleExtra`	String with extra title for the plot (appended after `title`).
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`labelVars`	Named character vector containing variable labels.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.

`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be: • compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`idVar`	String with variable containing subject ID.
`idLab`	String with label for `idVar`.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`watermark`	(optional) String with path to a file containing a watermark.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Lennart Tuijnder

See Also

Other visualizations of summary statistics for clinical data: `barplotClinData`(), `errorbarClinData`(), `plotCountClinData`(), `sunburstClinData`(), `treemapClinData`()

Examples

```
library(clinUtils)

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

## example of basic boxplot:

data <- subset(dataADaMCDISCP01$ADVS,
PARAMCD == "DIABP" & ANL01FL == "Y" &
AVISIT %in% c("Baseline", "Week 2", "Week 4", "Week 6", "Week 8")
)

## example of basic boxplot:

# With color var and facet:
boxplotClinData(
data = data,
xVar = "AVISIT",
yVar = "AVAL",
colorVar = "TRTA",
facetVar = "ATPT",
title = "Diastolic Blood Pressure distribution by actual visit and analysis timepoint",
yLab = "Actual value of the Diastolic Blood Pressure parameter (mmHg)",
labelVars = labelVars
)

# Control number of facet columns:
boxplotClinData(
data = data,
xVar = "AVISIT",
yVar = "AVAL",
colorVar = "TRTA",
facetVar = "ATPT",
ncol = 2,
title = "Diastolic Blood Pressure distribution by actual visit and analysis timepoint",
yLab = "Actual value of the Diastolic Blood Pressure parameter (mmHg)",
labelVars = labelVars
)

## Not run:

# Facet or color is optional:
boxplotClinData(
data = data,
xVar = "AVISIT",
yVar = "AVAL",
colorVar = "TRTA"
)

boxplotClinData(
data = data,
```

```
xVar = "AVISIT",
yVar = "AVAL",
facetVar = "ATPT"
)

# add caption & subtitle
boxplotClinData(
data = data,
xVar = "AVISIT",
yVar = "AVAL",
facetVar = "ATPT", ncol = 2,
colorVar = "TRTA",
title = "Diastolic Blood Pressure distribution",
subtitle = "By actual visit and analysis timepoint",
yLab = "Actual value of the Diastolic Blood Pressure parameter (mmHg)",
caption = "Summary statistics are computed internally.",
labelVars = labelVars
)


## End(Not run)
```

`buildBook` *Build the book*

Description

Build the book

Usage

```
buildBook(htmlFiles, verbose = TRUE)
```

Arguments

`htmlFiles`	character vector with path to HTML files
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.

Value

String with path to the front page of the report.

Author(s)

Laure Cougnaud

`checkAvailabilityMetadata` *Check availability of arguments in list*

Description

Check availability of arguments in list

Usage

```
checkAvailabilityMetadata(paramsList, subListName)
```

Arguments

`paramsList`	A named list.
`subListName`	String indicating which of the sublist names to check for existance.

Value

The content of the sublist. If not available, returns "Not Available".

`checkChapterParallel` *Check if a chapter is run internally in parallel or not.*

Description

This is identified via the 'parallel' parameter from the config file. If this parameter is not available in the config file (or the parameters are imported with an error), the chapter is considered to not be run in parallel.

Usage

```
checkChapterParallel(
  configFile,
  configDir = file.path(inputDir, "config"),
  inputDir = "."
)
```

Arguments

`configFile`	String with filename of the config file of interest in YAML format.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`inputDir`	String with input directory, working directory by default.

Value

Logical, if TRUE, the chapter is run in parallel (FALSE if not available).

`checkConfigFile` *Check a configuration file (in _YAML_format) based on a requirement file in JSON Schema format.*

Description

Check a configuration file (in _YAML_ format) based on a requirement file in JSON Schema format.

Usage

```
checkConfigFile(configFile, configSpecFile, configDir = "./config")
```

Arguments

`configFile`	path to the config file
`configSpecFile`	String with path to the file containing requirements in JSON Schema format.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.

Value

No returned value, an error message is printed in the console if the configuration file doesn't comply to the specified specifications.

Author(s)

Laure Cougnaud

`checkReportTitles` *Check report titles*

Description

Check uniqueness of report titles across the config files. If not unique titles are provided, an error is returned.

Usage

```
checkReportTitles(
  configFiles,
  configDir = file.path(inputDir, "config"),
  inputDir = "."
)
```

Arguments

`configFiles`	Character vector with config file names
`configDir`	String with directory with config files, ('config' by default)
`inputDir`	String with input directory, working directory by default.

Value

A named vector with the report titles and the corresponding config file

Author(s)

Michela Pasetto

See Also

Other clinical data reporting: `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_rep` `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`, `render_clinDataReviewRepo`

`checkTemplatesName` *Checks of config files template.*

Description

Check if the templates specified in the input config files don't originate from multiple sources (e.g. custom and R package via the parameter `templatePackage`). If so, the corresponding config files are not considered.

Usage

```
checkTemplatesName(
  configFiles,
  configDir = file.path(inputDir, "config"),
  inputDir = "."
)
```

Arguments

`configFiles`	Character vector with name or path of the config file(s).
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`inputDir`	String with input directory, working directory by default.

Value

Updated `configFiles`

Author(s)

Laure Cougnaud

`checkValueType` *Check if the specified* `valueType` *parameter can be passed to the* `branchvalues` *of the* `plot_ly` *treemap/sunburst visualizations.*

Description

If this parameter is set to 'total' and the sum of the counts of the the children nodes is not bigger than the parent node, an empty plot is created. In this case, this function set this parameter to: 'relative'.

Usage

```
checkValueType(data, vars, valueVar, valueType = "total", labelVars = NULL)
```

Arguments

`data`	Data.frame with data.
`vars`	Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).
`valueVar`	String with numeric variable of `data` containing the value to display.
`valueType`	String with type of values in `valueVar` (`branchvalues` of the `plot_ly`) function), among others: 'total' (default, only if sum(child) <= to parent) or 'relative'.
`labelVars`	Named character vector containing variable labels.

Value

If the condition is fullfilled: updated `valueType` and warning; otherwise input `valueType`.

Author(s)

Laure Cougnaud

`clinDataReview-common-args`

Common arguments for the functions of the clinDataReview package

Description

Common arguments for the functions of the clinDataReview package

Arguments

`data`	Data.frame with data.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.
`gg`	`ggplot` object.
`pl`	`plotly` object.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLab`	String with label for `xVar`.
`xLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the x-axis.
`yLab`	String with label for `xVar`.
`yLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the y-axis.
`xLim`, `yLim`	Numeric vector of length 2 with limits for the x/y axes.
`idVar`	String with variable containing subject ID.
`idLab`	String with label for `idVar`.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.
`lineVars`	List with parameters for the reference lines.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.

`pathExpand`	Logical, should the variable in `pathExpand` be included in a collapsible row or as hyperlink in the table? Should be TRUE for if multiple paths are included for each `idVar`, FALSE otherwise (by default).
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. The plot and table are linked when included in a Rmarkdown document: when clicking on an plot element, only the corresponding records are retained in the associated table; when some records are selected in the table, they are highlighted in the associated table.
`refLinePars`	(optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains: • aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`. • 'label': (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.
`labelVars`	Named character vector containing variable labels.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`title`	String with title for the plot.
`titleExtra`	String with extra title for the plot (appended after `title`).
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`colorVar`	(optional) String with color variable.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`keyVar`	String with unique key variable, identifying unique group for which the link between the table and the plot should be done.
`watermark`	(optional) String with path to a file containing a watermark.

Value

No return value, used for the documentation of the functions of the package.

`clinDataReview-common-args-report`

Common parameters for the clinical data reporting function

Description

Common parameters for the clinical data reporting function

Arguments

`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`configFile`	String with filename of the config file of interest in YAML format.
`inputDir`	String with input directory, working directory by default.
`outputDir`	String with output directory, ('report' by default).
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`extraDirs`	Character vector with extra directories required by the report, directory with external images. By default, the directories: 'figures', 'tables' and mentioned in the 'patientProfilePath' parameter of the general config file are included. All these folders should be available in `inputDir`.
`mdFile`	String with path of the Markdown file
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`nCores`	Integer containing the number of cores used to render the report (1 by default). If more than 1, two steps of the report creation are run in parallel across chapters: • the rendering of the Rmarkdown file to Markdown • the conversion from Markdown to HTML
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.

Value

No return value, used for the documentation of the clinical data reporting functions of the package.

`clinDataReview-common-args-summaryStatsVis`

Common arguments for the plotting functions summary statistics of the clinDataReview package

---

Description

Common arguments for the plotting functions summary statistics of the clinDataReview package

Arguments

`vars`	Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).
`varsLab`	Named character vector with labels for `vars`.
`valueVar`	String with numeric variable of `data` containing the value to display.
`valueLab`	String with label for the `valueVar` variable.
`valueType`	String with type of values in `valueVar` (`branchvalues` of the `plot_ly`) function), among others: 'total' (default, only if sum(child) <= to parent) or 'relative'.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be: • compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)

Value

No return value, used for the documentation of the plotting functions of summary statistics of the package.

`clinDataReview-templates`
Rmarkdown templates for clinical data

Description

Template reports with standard visualizations/tables available in the package are described here.

Details

For each template, required parameters are indicated in **bold**.

Value

No return value, used for the documentation of the Rmarkdown template reports contained in the package.

Parameter type

Please note that the type mentioned below corresponds to the type in the config file (in YAML/JSON format).
The mapping to R data type is as followed:

- string: character vector of length 1
- integer: integer vector of length 1
- array: vector/list without names
- object: list with names

Clinical data template for a visualization of count data : countsVisualizationTemplate

This report compute counts of variable(s) of interest (with the `inTextSummaryTable` package) and visualize them with a treemap and/or sunburst.
The following parameters are available:

- `template`: string set to: 'countsVisualizationTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted
- `reportTitle`: string, header title
- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.

- `dataFileName`: string or array, name of the data file(s) of interest. If multiple files are specified, the data are combined by rows ('row bind'), with a column: 'DATASET' containing the name of the file (in upper case and without extension) the data originate from.
- `dataProcessing`: (optional) array, data processing parameters, passed to processData
- `dataTotalFileName`: (optional) string, filename of the total dataset
- `dataTotalProcessing`: (optional) array, data processing parameters for 'dataTotalFileName', passed to processData
- `countVar`: string, variable of data to count on
- `parentVar`: (optional) string or array, parent variable(s) of the counting variable, used for grouping
- `colorVar`: (optional) string or object, numeric variable(s) to consider for coloring, named by count/parent variable if different for each variable
- `colorRange`: (optional) array of number(s) of length: minimum 2, maximum 2, range of the color variable for the visualization
- `loopingVar`: (optional) array or string, data variable(s) to loop over. Each group of the variable(s) is displayed in a separated section of the report.
- `loopingNMax`: (optional) integer, maximum number of elements of `loopingVar` to include in the report
- `loopingTotal`: (optional) boolean, should the total also be computed by `loopingVar` (TRUE by default)?
- `typePlot`: (optional) array or string of string(s) among: *'sunburst', 'treemap'*
- `startup`: (optional) array or string, R commands that should be run at the start of the report

Clinical data template to create a report division : divisionTemplate

This report includes a division, i.e. extra chapter, section of subsection in the report.
The following parameters are available:

- `template`: string set to: 'divisionTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted
- `reportTitle`: string, header title
- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `content`: (optional) string, any content that should be included after the title

Clinical data template to create a listing : listingTemplate

This report displays a listing of the variables and data of interest, displayed in an interactive table. This table can contains comparison with a previous batch ('comparisonTable' parameters).
The following parameters are available:

- `template`: string set to: 'listingTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted
- `reportTitle`: string, header title
- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `dataFileName`: string or array, name of the data file(s) of interest. If multiple files are specified, the data are combined by rows ('row bind'), with a column: 'DATASET' containing the name of the file (in upper case and without extension) the data originate from.
- `dataProcessing`: (optional) array, data processing parameters, passed to processData
- `dataTotalFileName`: (optional) string, filename of the total dataset
- `dataTotalProcessing`: (optional) array, data processing parameters for 'dataTotalFileName', passed to processData
- `tableParams`: (optional) object, parameters to create the table, passed to tableClinData
- `comparisonTableType`: (optional) string among: *'none', 'newData-diff-interactive', 'table-comparison-interactive'* , output type of the comparison table
- `comparisonTableParams`: (optional) object, parameters for the comparison table, passed to compareTables
- `loopingVar`: (optional) array or string, data variable(s) to loop over. Each group of the variable(s) is displayed in a separated section of the report.
- `loopingNMax`: (optional) integer, maximum number of elements of `loopingVar` to include in the report
- `listingDocx`: (optional) boolean, export listing to Word

Clinical data template for the creation of patient profiles : patientProfilesTemplate

This report creates the specified patient profiles (with the `patientProfilesVis` package) by subject, and export them to a specified directory.
The following parameters are available:

- `template`: string set to: 'patientProfilesTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted

- `reportTitle`: string, header title
- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `createPatientProfiles`: (optional) boolean, Should the patient profiles be created or only loaded from a previous execution?
- `patientProfilesGeneralParams`: (optional) object, set of parameters used for all patient profiles modules. These parameters are passed to all `subjectProfile[]Plot` functions.
- `patientProfilesParams`: array of object(s)
  The following parameters are available:
  - `typePlot`: string among: *'text', 'line', 'interval', 'event'* , plot type, used to get the appropriate plot module function:
    - 'text': subjectProfileTextPlot
    - 'line': subjectProfileLinePlot
    - 'interval': subjectProfileIntervalPlot
    - 'event': subjectProfileEventPlot
  - `dataFileName`: string, name of the data file of interest
  - `dataProcessing`: (optional) array, data processing parameters, passed to processData
  - `plotParams`: object, parameters for the plotting function. Parameters depending on the dataset of interest can be specified as:
    `[parameterName]: !r-lazy [dataI]`

  , parameters for each patient profile module
- `patientProfilesCreateReportParams`: (optional) object, parameters for the creation of the patient profile report(s), passed to createSubjectProfileReport
- `tableParams`: (optional) object, parameters specifying a table containing data of interest and links to created patient profiles
- `startup`: (optional) array or string, R commands that should be run at the start of the report

Clinical data generic template for visualization : plotTemplate

This report visualizes input data with a function of the clinical data review package. The data can be compared to the data of a previous batch, in the table associated to the plot ('comparisonTable' parameters).Summary statistics can be computed optionally and included in the plot (see 'tableParams' parameter).
The following parameters are available:

- `template`: string set to: 'plotTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted

- `reportTitle`: string, header title
- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `dataFileName`: string or array, name of the data file(s) of interest. If multiple files are specified, the data are combined by rows ('row bind'), with a column: 'DATASET' containing the name of the file (in upper case and without extension) the data originate from.
- `dataProcessing`: (optional) array, data processing parameters, passed to processData
- `plotFunction`: string among: *'timeProfileIntervalPlot', 'scatterplotClinData', 'boxplotClinData'* , plotting function of the package to consider
- `plotParams`: object, parameters for the plotting function. Parameters depending on the dataset of interest can be specified as:
  `[parameterName]: !r-lazy [dataI]`
- `tableParams`: (optional) object, parameters for a summary table, passed to computeSummaryStatisticsTable
  Summary statistics are computed and merged as extra columns available for the plot data.
- `tableProcessing`: (optional) array, data processing parameters for the summary table, passed to processData
- `comparisonTableType`: (optional) string among: *'none', 'newData-diff'* , output type of the comparison table. If specified, an additional column: 'Comparison Type' is included in the table attached to the plot.
- `comparisonTableParams`: (optional) object, parameters for the comparison table, passed to compareTables. If referenceVars is not specified, all variables displayed in the plot are used.
- `loopingVar`: (optional) array or string, data variable(s) to loop over. Each group of the variable(s) is displayed in a separated section of the report.
- `loopingNMax`: (optional) integer, maximum number of elements of `loopingVar` to include in the report
- `startup`: (optional) array or string, R commands that should be run at the start of the report

Clinical data template for visualization of summarized data : summaryPlotTemplate

This report summarizes the data of interest (with the `inTextSummaryTable` package) and visualize it with any clinical data review plot function.
The following parameters are available:

- `template`: string set to: 'summaryPlotTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted
- `reportTitle`: string, header title

- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `dataFileName`: string or array, name of the data file(s) of interest. If multiple files are specified, the data are combined by rows ('row bind'), with a column: 'DATASET' containing the name of the file (in upper case and without extension) the data originate from.
- `dataProcessing`: (optional) array, data processing parameters, passed to processData
- `dataTotalFileName`: (optional) string, filename of the total dataset
- `dataTotalProcessing`: (optional) array, data processing parameters for 'dataTotalFileName', passed to processData
- `tableParams`: object, parameters to summarize the data in a table, passed to computeSummaryStatisticsTable
- `tableProcessing`: (optional) array, data processing parameters for the summary table, passed to processData
- `plotFunction`: string among: *'timeProfileIntervalPlot', 'scatterplotClinData', 'sunburstClinData', 'treemapClinData', 'barplotClinData', 'errorbarClinData'* , plotting function to visualize summary data
- `plotParams`: object, parameters for the plotting function. Parameters depending on the dataset of interest can be specified as:
  `[parameterName]: !r-lazy [summaryTableI]`
- `loopingVar`: (optional) array or string, data variable(s) to loop over. Each group of the variable(s) is displayed in a separated section of the report.
- `loopingNMax`: (optional) integer, maximum number of elements of `loopingVar` to include in the report
- `loopingTotal`: (optional) boolean, should the total also be computed by `loopingVar` (TRUE by default)?
- `startup`: (optional) array or string, R commands that should be run at the start of the report

Clinical data template for a summary table of the data : summaryTableTemplate

This report summarizes the data of interest (with the `inTextSummaryTable` package). This table is displayed with an interactive table in the report, and exported to a docx file. This table can be compared to a summary table of a previous batch ('comparisonTable' parameters).
The following parameters are available:

- `template`: string set to: 'summaryTableTemplate.Rmd', name of the template report
- `templatePackage`: string set to: 'clinDataReview', package from which the template should be extracted
- `reportTitle`: string, header title

- `reportTitleLevel`: (optional) integer, header level, 1 by default (1: 'chapter', '2': 'section', '3': subsection, ...)
- `parallel`: (optional) boolean, does this chapter use parallel execution? If the entire report is run in parallel, this ensures that this specific chapter is created outside of the report parallelization.
- `split_by`: (optional) integer of length: minimum 0, maximum 7 or string among: *'none', 'chapter', 'section'* , split the chapter at the specified level: 1 (for 'chapter'), 2 (for 'section') until 7. This overwrites the 'split_by' parameter defined in the output format of the report.
- `dataFileName`: string or array, name of the data file(s) of interest. If multiple files are specified, the data are combined by rows ('row bind'), with a column: 'DATASET' containing the name of the file (in upper case and without extension) the data originate from.
- `dataProcessing`: (optional) array, data processing parameters, passed to processData
- `dataTotalFileName`: (optional) string, filename of the total dataset
- `dataTotalProcessing`: (optional) array, data processing parameters for 'dataTotalFileName', passed to processData
- `tableParams`: object, parameters to summarize the data in a table, passed to computeSummaryStatisticsTable .Parameters depending on the dataset of interest can be specified as: `[parameterName]: !r-lazy [dataI]`.
- `tableParamsDocx`: (optional) object, parameters to format the table to the docx format, passed to exportSummaryStatisticsTable
- `tableParamsDT`: (optional) object, parameters to format the table to the DT interactive table included in the report, passed to exportSummaryStatisticsTable
- `comparisonTableType`: (optional) string among: *'none', 'table-comparison-interactive', 'newData-diff', 'table-combine'* , output type of the comparison table:
  - 'table-comparison-interactive': the summary statistics are computed for the old and new batch separately, and the statistics are compared
  - 'newData-diff': the differences at a record level between the new and old batch are first identified, and then summarized
  - 'table-combine': the summary tables are computed for the old and new batch and displayed next to each other in a combined table
- `comparisonTableParams`: (optional) object, parameters for the comparison table, passed to compareTables. By default, statistics variables are compared across row and column elements.
- `loopingVar`: (optional) array or string, data variable(s) to loop over. Each group of the variable(s) is displayed in a separated section of the report.
- `loopingNMax`: (optional) integer, maximum number of elements of `loopingVar` to include in the report
- `loopingTotal`: (optional) boolean, should the total also be computed by `loopingVar` (TRUE by default)?

`collapseHtmlContent` *Function to create collapsible HTML content*

Description

Please note that the button is of class: ’hideshow’, defined in the ’input.hideshow.js’ js file included in the package.

Usage

```
collapseHtmlContent(input, title = "Click to show or hide")
```

Arguments

`input`	Object to be collapse, e.g. datatable.
`title`	String with button title.

Value

`tag` object

Author(s)

Laure Cougnaud

`combineButtonsAndPlot` *Combine select box(es) and the plot*

Description

Combine select box(es) and the plot

Usage

```
combineButtonsAndPlot(x)
```

Arguments

`x`	Object of class `clinDataReview`

Value

`x` object:

- with the `plot` element containing a combination of the `buttons` and the `plot`
- without the `buttons` element

`convertMdToHtml` *Convert the Md file for a specific chapter to html*

Description

Convert the Md file for a specific chapter to html

Usage

```
convertMdToHtml(
  mdFile,
  configFile = NULL,
  indexPath = "index.Rmd",
  intermediateDir = "./interim",
  outputDir = "./report",
  setTitle = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`mdFile`	String with path of the Markdown file
`configFile`	String with filename of the config file of interest in YAML format.
`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`outputDir`	String with output directory, ('report' by default).
`setTitle`	Logical (TRUE by default), should the title be set to the document? If so, the pandoc metadata option: 'pagetitle' is set to: base file name of `mdFile`.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.
`...`	Arguments passed to `renderFile`

Value

No returned value, the files in the `intermediateDir` are converted to HTML

Author(s)

Laure Cougnaud

`convertReportToAspx` *Convert report format from .html to .aspx*

Description

Report files generated as output of `render_clinDataReviewReport` are converted from .html to to .aspx format by changing extensions and cross-links of all files within the directory `reportDir`. This allows for deployment on SharePoint.

Usage

```
convertReportToAspx(reportDir = ".")
```

Arguments

`reportDir`	String for the path to the directory where the clinical data reports are stored, defaults to current directory

Value

no return value, files in directory are modified

`countNLines` *Count number of lines in a vector*

Description

Count number of lines in a vector

Usage

```
countNLines(x)
```

Arguments

`x`	Character vector.

Value

Integer vector of length `x` with number of lines

Author(s)

Laure Cougnaud

Examples

```
clinDataReview:::countNLines(x = c("A\nB", "blabla", "This\nis\na\nsentence."))
```

`createClinDataReviewReportSkeleton` *Create the skeleton of a report*

Description

Creates the skeleton of a report to start running the analyses.

Usage

```
createClinDataReviewReportSkeleton(dir = ".")
```

Arguments

`dir`	String with the path of the directory where the skeleton should be created. The current working directory is used as default.

Details

This function is meant to get familiar with the use of the package and the necessary files to create a report.

It will create a ready-to-use report with example data from the `clinUtils` package. After getting use to the file structure, the user can substitute the example data with custom data sets and add specific configuration files.

Value

The files to run a report are written in the specified directory. To run the report, the user can call the `render_clinDataReviewReport`.

`createExampleMetadata` *Create an example metadata file*

Description

Create an example of metadata file for the `createClinDataReviewReportSkeleton`.

Usage

```
createExampleMetadata(dir)
```

Arguments

`dir`	String, path to the directory.

Value

Nothing, the example metadata file is created in the specified directory.

`createMainConfigSkeleton` *Create the config file for the skeleton*

Description

This function creates the main config file for the `createClinDataReviewReportSkeleton` with the directory where the data are stored.

Usage

```
createMainConfigSkeleton(dir, dirData)
```

Arguments

`dir`	String, path to the directory.
`dirData`	String, path to the directory of the data.

Value

No return value, a file _config.yml_ is created in the specified directory.

`createOutputYaml` *Create a output YAML file*

Description

This file containing the contents of the `output` field of the YAML header of a Markdown file. It can be passed to the `output_yaml` parameter of the `render` function.

Usage

```
createOutputYaml(indexPath, outputDir)
```

Arguments

`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`outputDir`	String with output directory, ('report' by default).

Value

String with file to the `_output.yml` file in a temporary folder.

`createPatientProfileVar`

Create link to patient profile

Description

Create a link to a patient profile directory (where the patient profile files are saved) by adding an extra column with the link in the data. The path to the patient profile is built as: [patientProfilePath]/subjectProfile-[subjectID].pdf, where '/' are replaced with '-' in the subject identifier (`subjectVar`).

Usage

```
createPatientProfileVar(
  data,
  patientProfilePath,
  subjectVar = "USUBJID",
  checkExist = TRUE
)
```

Arguments

`data`	a data.frame
`patientProfilePath`	string indicating the directory where the patient profiles are stored.
`subjectVar`	string indicating which column in the data represents the unique subject identifier, "USUBJID" by default.
`checkExist`	Logical, if TRUE (by default) the `patientProfilePath` is checked for existence, and an error is returned if this directory doesn't exist.

Value

A data.frame with two extra columns: `patientProfilePath` and `patientProfileLink` with the path to the patient profile and an hyperlink to it, respectively.

Author(s)

Michela Pasetto

Examples

```
# Typical CDISC dataset contains universal subject ID (USUBJID)
data <- data.frame(USUBJID = c("subj1", "subj2", "subj3"))
dataWithPatientProfileVar <- createPatientProfileVar(
  data = data,
  patientProfilePath = "pathProfiles",
  checkExist = FALSE
)
# path and HTML link are included in the output dataset
head(dataWithPatientProfileVar[, c("USUBJID", "patientProfilePath", "patientProfileLink")])
```

`createRedirectPage` *Create a redirect page*

Description

Create an html page that redirects to the "1-introduction.html" page of the clinical data report available in a directory. See output from `render_clinDataReviewReport`.

Usage

```
createRedirectPage(redirectPage = "report.html", dir = "report_dependencies")
```

Arguments

`redirectPage`	String with the path of the html file that redirects to the "1-introduction.html" page of the report.
`dir`	String for the path where the "1-introduction.html" is stored.

Value

The html file is created.

`createTemplateDoc` *Create documentation for clinical data template reports available in the 'template' folder of the package.*

Description

If a JSON schema file available, the information relative to the template is extracted from this file with the function `JSONSchToRd`.

Usage

```
createTemplateDoc(
  templatePath = system.file("template", package = "clinDataReview")
)
```

Arguments

`templatePath`	string with path where the template Rmd reports and associated JSON schema files are stored, by default path of the installed version of the package. This parameter is only for expert use of the package.

Value

Character vector with Rd code containing description for all template documents.

Author(s)

Laure Cougnaud

References

JSON schema specification

`errorbarClinData` *Interactive plot of confidence interval/error interval of clinical data.*

Description

This plot is designed to display summary statistics of a continuous variable with (confidence) intervals.
The intervals are either displayed:

- vertically if `yErrorVar` is specified
- horizontally if `xErrorVar` is specified

Error bars can visualized by group, via the color variable parameter.
Different symbols are set for each central point of the error bar via the shape variable parameter.

Usage

```
errorbarClinData(
  data,
  xVar,
  xLab = getLabelVar(xVar, labelVars = labelVars),
  xLabVar = NULL,
  yVar,
  yLab = getLabelVar(yVar, labelVars = labelVars),
  yLabVar = NULL,
  yErrorVar = NULL,
  yErrorLab = getLabelVar(yErrorVar, labelVars = labelVars),
  xErrorVar = NULL,
  xErrorLab = getLabelVar(xErrorVar, labelVars = labelVars),
  xLabVars = NULL,
  xAxisLab = paste(c(xLab, xErrorLab), collapse = " and "),
  yAxisLab = paste(c(yLab, yErrorLab), collapse = " and "),
  colorVar = NULL,
  colorLab = getLabelVar(colorVar, labelVars = labelVars),
  colorPalette = NULL,
  shapeVar = NULL,
  shapeLab = getLabelVar(shapeVar, labelVars = labelVars),
  shapePalette = NULL,
  size = 6,
 title = paste(c(paste(yAxisLab, "vs", xAxisLab), titleExtra), collapse = "<br>"),
```

```
  titleExtra = NULL,
  subtitle = NULL,
  caption = NULL,
  labelVars = NULL,
  mode = "markers",
  legendPosition = "bottom",
  width = NULL,
  height = NULL,
  pathVar = NULL,
  pathLab = getLabelVar(pathVar, labelVars = labelVars),
  hoverVars,
  hoverLab,
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  selectVars = NULL,
  selectLab = getLabelVar(selectVars, labelVars = labelVars),
  table = FALSE,
  tableVars,
  tableLab,
  tableButton = TRUE,
  tablePars = list(),
  watermark = NULL,
  verbose = FALSE
)
```

Arguments

`data`	Data.frame with data.
`xVar`	String with column of `data` containing x-variable.
`xLab`	String with label for `xVar`.
`xLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the x-axis.
`yVar`	String with column of `data` containing y-variable.
`yLab`	String with label for `xVar`.
`yLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the y-axis.
`xErrorVar`, `yErrorVar`	String with variable of `data` containing the width of the interval (from the center of the interval) for horizontal or vertical intervals.
`xErrorLab`, `yErrorLab`	String with labels for `xErrorVar`/`yErrorVar` variables.
`xLabVars`	(vertical error bars) Character vector with variable(s) to be displayed as the labels of the ticks in the x-axis. By default, `xVar` is displayed. In case the variable(s) contain different elements by `xVar`, they are combined and displayed below each other.
`xAxisLab`, `yAxisLab`	Label for the x/y-axis.

`colorVar`	(optional) String with color variable.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`shapeVar`	(optional) String with shape variable.
`shapeLab`	String with label for `shapeVar`.
`shapePalette`	(optional) Named character vector with shape palette, `clinShapes` by default.
`size`	Integer with size of markers in pixels, 6 by default.
`title`	String with title for the plot.
`titleExtra`	String with extra title for the plot (appended after `title`).
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`labelVars`	Named character vector containing variable labels.
`mode`	String with the mode of the plot, 'markers' by default, so only data points are displayed. This can also be set to 'lines' to include a line connecting the center of the error bars instead; or 'lines+markers' to include both a marker and a line. See `mode` attribute for plotly scatter.
`legendPosition`	String with position of the legend, among: 'top'/'left'/'bottom'/'right', 'bottom' by default.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be: • compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.
`id`	String with general id for the plot:

- 'id' is used as group for the `SharedData`
- 'button:[id]' is used as button ID if `table` is TRUE

If not specified, a random id, as 'plotClinData[X]' is used.

`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`watermark`	(optional) String with path to a file containing a watermark.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other visualizations of summary statistics for clinical data: `barplotClinData()`, `boxplotClinData()`, `plotCountClinData()`, `sunburstClinData()`, `treemapClinData()`

Examples

```
library(clinUtils)

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

## Summary plot with vertical error bars

dataVSDIABP <- subset(dataADaMCDISCP01$ADVS,
PARAMCD == "DIABP" & ANL01FL == "Y" &
AVISIT %in% c("Baseline", "Week 2", "Week 4", "Week 6", "Week 8")
```

```
)

# compute summary statistics by visit
if (requireNamespace("inTextSummaryTable", quietly = TRUE)) {

summaryTableVSDIABP <- inTextSummaryTable::computeSummaryStatisticsTable(
data = dataVSDIABP,
rowVar = c("AVISIT", "ATPT"),
var = "AVAL",
stats = inTextSummaryTable::getStats(c("n", "Mean", "SE")),
labelVars = labelVars
)
dataPlot <- subset(summaryTableVSDIABP, !isTotal)

errorbarClinData(
data = dataPlot,
xVar = "AVISIT",
colorVar = "ATPT",
# use non-rounded statistics for the plot
yVar = "statMean", yErrorVar = "statSE",
yLab = "Mean", yErrorLab = "Standard Error",
# include lines connecting the error bars
mode = "markers+lines",
labelVars = labelVars
)

# add number of subjects in labels
dataPlot$nSubj <- with(dataPlot, paste0("N=", n))
errorbarClinData(
data = dataPlot,
xVar = "AVISIT",
xLabVars = c("AVISIT", "nSubj"),
colorVar = "ATPT",
yVar = "statMean", yLab = "Mean",
yErrorVar = "statSE", yErrorLab = "Standard error",
mode = "markers+lines",
title = paste("Diastolic Blood Pressure summary profile by actual visit",
"and analysis timepoint"),
labelVars = labelVars
)

## Add a selection box
if(interactive()){
  summaryTable <- inTextSummaryTable::computeSummaryStatisticsTable(
    data = subset(dataADaMCDISCP01$ADVS,
      ANL01FL == "Y" &
      AVISIT %in% c("Baseline", "Week 2", "Week 4", "Week 6", "Week 8")
    ),
    rowVar = c("PARAM", "AVISIT", "ATPT"),
    var = "AVAL",
    stats = inTextSummaryTable::getStats(c("Mean", "SE")),
    labelVars = labelVars
  )
```

```
  dataPlot <- subset(summaryTable, !isTotal)
  errorbarClinData(
    data = dataPlot,
    xVar = "AVISIT",
    colorVar = "ATPT",
    yVar = "statMean", yLab = "Mean",
    yErrorVar = "statSE", yErrorLab = "Standard error",
    mode = "markers+lines",
    title = paste("Lab parameters summary profile by actual visit",
      "and analysis timepoint"),
    labelVars = labelVars,
    selectVars = "PARAM"
  )
}

## Summary plot with horizontal error bars

# Data of interest: ratio from baseline at week 16
dataLBW8 <- subset(dataADaMCDISCP01$ADLBC, grepl("Week 8", AVISIT))
# compute ratio from baseline
dataLBW8$R2BASE <- with(dataLBW8, AVAL/BASE)
dataLBW8 <- subset(dataLBW8, !is.na(R2BASE))
# Order actual treatments
dataLBW8$TRTA <- with(dataLBW8, reorder(TRTA, TRTAN))

# compute summary statistics of the ratio per baseline per parameter
summaryTableLBW8 <- inTextSummaryTable::computeSummaryStatisticsTable(
data = dataLBW8,
var = "R2BASE",
rowVar = "PARAM",
colVar = "TRTA",
stats = inTextSummaryTable::getStats(x = dataLBW8$R2BASE, type = c("n", "Median", "SD"))
)
dataPlot <- subset(summaryTableLBW8, !isTotal)
# extract direction of ratio
dataPlot$dir <- factor(
ifelse(dataPlot$statMedian >= 1, "Increase", "Decrease"),
levels = c("Decrease", "Increase")
)
# compute relative ratio (percentage above 1)
dataPlot$statMedianRelative <- with(dataPlot,
ifelse(statMedian < 1, 1/statMedian, statMedian)
)
# order based on mean relative ratio across treatment arms
params <- names(sort(with(dataPlot, tapply(statMedianRelative, PARAM, mean))))
dataPlot$PARAM <- factor(dataPlot$PARAM, levels = params)
errorbarClinData(
data = dataPlot,
xVar = "statMedianRelative", xErrorVar = "statSD",
xLab = "Median", xErrorLab = "Standard deviation",
xAxisLab = "Relative ratio from baseline (Median +- SD)",
yVar = "PARAM",
colorVar = "TRTA",
```

```
shapeVar = "dir", shapeLab = "Direction of ratio",
shapePalette = c(`Decrease` = 25, `Increase` = 24),
size = 10,
labelVars = labelVars,
title = "Summary ratio from baseline at week 8 by treatment"
)

}
```

`exportSessionInfoToMd` *Combine all session informations across all clinical data reports and export them into a dedicated Markdown document*

Description

Combine all session informations across all clinical data reports and export them into a dedicated Markdown document

Usage

```
exportSessionInfoToMd(
  sessionInfos,
  intermediateDir = "interim",
  logFile = NULL,
  ...
)
```

Arguments

`sessionInfos`	List with `sessionInfo` objects
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`...`	Any parameters passed to `renderFile`, for expert use only.

Value

String with path to Markdown file containing the session information, NULL if no session information(s) are provided.

Author(s)

Laure Cougnaud

`filterData` *Filter a dataset based on specified filters.*

Description

A dataset can be filtered:

- on a specific `value` of interest
- on a function of a variable (`valueFct` parameter), e.g. maximum of the variable)
- to retain only non missing values of a variable (`keepNA` set to `FALSE`)
- by groups (`varsBy` parameter)

Note that by default, missing values in the filtering variable are retained (which differs from the default behaviour in R). To filter missing records, please set the `keepNA` parameter to `FALSE`.

Usage

```
filterData(
  data,
  filters,
  keepNA = TRUE,
  returnAll = FALSE,
  verbose = FALSE,
  labelVars = NULL,
  labelData = "data"
)
```

Arguments

`data`	Data.frame with data.
`filters`	Unique filter or list of filters. Each filter is a list containing: • 'var': String with variable from `data` to filter on. • 'value': (optional) Character vector with values from `var` **to consider/keep**. • 'valueFct': (optional) Function (or string with this function) to be applied on `var` to extract value to consider. For example, `valueFct = max` will extract the records with the maximum value of the variable. • 'op': (optional) String with operator used to retain records from `value`. If not specified, the inclusion operator: '%in%' is considered, so records with `var` in `value` are retained. • 'rev': (optional) Logical, if TRUE (FALSE by default), filtering condition based on `value`/`valueFct` is reversed. • 'keepNA': (optional) Logical, if TRUE (by default), missing values in `var` are retained. If not specified, `keepNA` general parameter is used.

- ’varsBy’: (optional) Character vector with variables in `data` containing groups to filter by
- ’postFct’: (optional) Function (or string with this function) with post-processing applied on the results of the filtering criteria (TRUE/FALSE for each record). This function should return TRUE/FALSE (for each record or for all considered records).
  For example, ’`postFct = any, varsBy = "group"`’ retains all groups which contain at least one record that fulfills the criteria.
- ’varNew’: (optional) String with name of a new variable containing the results of the filtering criteria (as TRUE/FALSE).
- ’labelNew’: (optional) String with label for the `varNew` variable.

If a list of filters is specified, the different filters are **independently executed on the entire dataset to identify the records to retain for each filtering condition.**
The resulting selections are combined with a `Logic` operator (’&’ by default, i.e. ’AND’ condition). A custom logic operator can be specified between the lists describing the filter, for example:
`list(list(var = "SEX", value = "F"), "&",list(var = "COUNTRY", value = "DEU"))`.

`keepNA`	Logical, if TRUE (by default) missing values in `var` are retained. If set to FALSE, missing values are ignored for all filters. The specification within `filters` prevails on this parameter.
`returnAll`	Logical: • if FALSE (by default): the `data` for only the filtered records is returned. • if TRUE: the full `data` is returned. Records are flagged based on the `filters` condition, in a new column: `varNew` (if specified), or ’keep’ otherwise; containing TRUE if the record fulfill all conditions, FALSE otherwise
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.
`labelVars`	Named character vector containing variable labels.
`labelData`	(optional) String with label for input `data`, that will be included in progress messages.

Value

If `returnAll`

- is `FALSE`: `data` filtered with the specified filters
- is `TRUE`: `data` with the additional column: `keep` or `varNew` (if specified), containing `TRUE` for records which fulfill the specified condition(s) and `FALSE` otherwise.

The output contains the additional attribute: `msg` which contains a message describing the filtered records.

Author(s)

Laure Cougnaud

Examples

```
library(clinUtils)

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

dataDM <- dataADaMCDISCP01$ADSL

## single filter

# filter with inclusion criteria:
filterData(
data = dataDM,
filters = list(var = "SEX", value = "M"),
# optional
labelVars = labelVars, verbose = TRUE
)

# filter with non-inclusion criteria
filterData(
data = dataDM,
filters = list(var = "SEX", value = "M", rev = TRUE),
# optional
labelVars = labelVars, verbose = TRUE
)

# filter based on inequality operator
filterData(
data = dataDM,
filters = list(var = "AGE", value = 75, op = "<="),
# optional
labelVars = labelVars, verbose = TRUE
)

# missing values are retained by default!
dataDMNA <- dataDM
dataDMNA[1 : 2, "AGE"] <- NA
filterData(
data = dataDMNA,
filters = list(var = "AGE", value = 75, op = "<="),
# optional
labelVars = labelVars, verbose = TRUE
)

# filter missing values on variable
filterData(
data = dataDMNA,
filters = list(var = "AGE", value = 75, op = "<=", keepNA = FALSE),
# optional
labelVars = labelVars, verbose = TRUE
)
```

```
# retain only missing values
filterData(
data = dataDMNA,
filters = list(var = "AGE", value = NA, keepNA = TRUE),
# optional
labelVars = labelVars, verbose = TRUE
)

# filter missing values
filterData(
data = dataDMNA,
filters = list(var = "AGE", keepNA = FALSE),
# optional
labelVars = labelVars, verbose = TRUE
)


## multiple filters

# by default the records fulfilling all conditions are retained ('AND')
filterData(
data = dataDM,
filters = list(
list(var = "AGE", value = 75, op = "<="),
list(var = "SEX", value = "M")
),
# optional
labelVars = labelVars, verbose = TRUE
)

# custom operator:
filterData(
data = dataDM,
filters = list(
list(var = "AGE", value = 75, op = "<="),
"|",
list(var = "SEX", value = "M")
),
# optional
labelVars = labelVars, verbose = TRUE
)

# filter by group

# only retain adverse event records with worst-case severity
dataAE <- dataADaMCDISCP01$ADAE
dataAE$AESEV <- factor(dataAE$AESEV, levels = c("MILD", "MODERATE", "SEVERE"))
dataAE$AESEVN <- as.numeric(dataAE$AESEV)
nrow(dataAE)
dataAEWorst <- filterData(
data = dataAE,
filters = list(
var = "AESEVN",
```

```
valueFct = max,
varsBy = c("USUBJID", "AEDECOD"),
keepNA = FALSE
),
# optional
labelVars = labelVars, verbose = TRUE
)
nrow(dataAEWorst)

# post-processing function
# keep subjects with at least one severe AE:
dataSubjectWithSevereAE <- filterData(
  data = dataAE,
  filters = list(
    var = "AESEV",
    value = "SEVERE",
    varsBy = "USUBJID",
    postFct = any
  ),
  # optional
  labelVars = labelVars, verbose = TRUE
)

# for each laboratory parameter: keep only subjects which have at least one
# measurement classified as low or high
dataLB <- subset(dataADaMCDISCP01$ADLBC, !grepl("change", PARAM))
dataLBFiltered <- filterData(
  data = dataLB,
  filters = list(
    var = "LBNRIND",
    value = c("LOW", "HIGH"),
    varsBy = c("PARAMCD", "USUBJID"),
    postFct = any
  ),
  # optional
  labelVars = labelVars, verbose = TRUE
)
```

`filterDataSingle` *Filter data for a single filter*

Description

Filter data for a single filter

Usage

```
filterDataSingle(
  data,
  filters,
```

```
  keepNA = TRUE,
  returnAll = FALSE,
  labelVars = NULL,
  labelData = "data"
)
```

Arguments

`data`	Data.frame with data.
`filters`	Unique filter or list of filters.
`keepNA`	Logical, if TRUE (by default) missing values in `var` are retained. If set to FALSE, missing values are ignored for all filters. The specification within `filters` prevails on this parameter.
`returnAll`	Logical: • if FALSE (by default): the `data` for only the filtered records is returned. • if TRUE: the full `data` is returned. Records are flagged based on the `filters` condition, in a new column: `varNew` (if specified), or 'keep' otherwise; containing TRUE if the record fulfill all conditions, FALSE otherwise
`labelVars`	Named character vector containing variable labels.
`labelData`	(optional) String with label for input `data`, that will be included in progress messages.

Value

Updated `data` with attributes:

- 'labelVars': input `labelVars` with any new variables if `labelNew` is specified.
- 'msg': message describing the filtering process
- 'warn': warning describing the filtering process

Author(s)

Laure Cougnaud

`forceParams` *Force the evaluation of the parameters from config file.*

Description

This function is only useful if some parameters should be lazy-evaluated in the report. These parameters should have the class: `r-lazy`. A typical use case is a parameter that consists of a R expression depending on objects created in a template report (typically `data`).
Parameters are searched in the environment in which this function is called from.

Usage

```
forceParams(params)
```

Arguments

`params`	List of parameters as obtained via the `getParamsFromConfig` function.

Value

Input parameter list, with object(s) of class `r-lazy` evaluated.

Author(s)

Laure Cougnaud

See Also

getParamsFromConfig

Other clinical data reporting: `checkReportTitles()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`, `render_clinDataReviewReport()`

Examples

```
data <- mtcars
params <- list(label = "Cars dataset", nrow = structure("nrow(data)", class = "r-lazy"))
str(params)
str(forceParams(params))
```

`formatDataForPlotClinData`

Format data for interactive plot for clinical data

Description

Format data for interactive plot for clinical data

Usage

```
formatDataForPlotClinData(
  data,
  hoverVars = NULL,
  hoverLab = getLabelVar(hoverVars, labelVars = labelVars),
  hoverByVar = NULL,
  keyVar = NULL,
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  labelVars = NULL
)
```

Arguments

`data`	Data.frame with data.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.
`hoverByVar`	Character vector with variables identifying unique elements in the plot, usually x, y, facet variables. These variables are used to identify records with the same position in the plot, their information are combined in the hover.
`keyVar`	String with unique key variable, identifying unique group for which the link between the table and the plot should be done.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`labelVars`	Named character vector containing variable labels.

Value

`SharedData` object containing the `data`, with an extra column: 'hover' with the combined info from `hoverVars`, and the key defined as `keyVar` and group as `id`.

Author(s)

Laure Cougnaud

`formatHoverText` *Format hover text for use in plotly interactive plots. The labels are wrapped to multiple lines if exceed the width of the plotly hover box, e.g. in case labels for points with same x/y coordinates overlap, and corresponding labels are truncated.*

Description

Format hover text for use in plotly interactive plots. The labels are wrapped to multiple lines if exceed the width of the plotly hover box, e.g. in case labels for points with same x/y coordinates overlap, and corresponding labels are truncated.

Usage

```
formatHoverText(x, label, width = 50)
```

Arguments

`x`	Vector with hover text information.
`label`	Label for the variable
`width`	Integer, number of characters at which the hover text should be cut at to multiple lines.

Value

String with formatted hover label.

Author(s)

Laure Cougnaud

`formatPathDateInfoMetadata`

Format the info on paths from metadata

Description

Format the info on paths from metadata

Usage

```
formatPathDateInfoMetadata(summaryInfo, namesInfo)
```

Arguments

`summaryInfo`	matrix, see output from `getMetadata`.
`namesInfo`	Named vector to rename the final output.

Value

A kable object, to be printed.

`formatPlotlyClinData` *Format interactive plot, with possibility to download patient profiles on a click event.*

Description

Format interactive plot, with possibility to download patient profiles on a click event.

Usage

```
formatPlotlyClinData(
  pl,
  data,
  idVar = "USUBJID",
  pathVar = NULL,
  pathDownload = TRUE,
  idFromDataPlot = FALSE,
  idVarPlot = "key",
  labelVarPlot = NULL,
  highlightOn = "plotly_click",
  highlightOff = "plotly_doubleclick",
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  selectVars = NULL,
  selectLab = getLabelVar(selectVars, labelVars = labelVars),
  keyVar = NULL,
  keyHighlightBox = FALSE,
  labelVars = NULL,
  verbose = FALSE
)
```

Arguments

`pl`	`plotly` object.
`data`	Data.frame with data.
`idVar`	String with variable of `data` containing plot element.
`pathVar`	String with variable of `data` containing path to a subject-specific report (e.g. patient profiles).
`pathDownload`	Logical, if TRUE (by default) the subject-specific report(s) are downloaded in a zip compressed file. If FALSE (only available if unique report per `idVarPlot`), each report is opened in a new window.
`idFromDataPlot`	Logical, if TRUE (by default) `idVarPlot` is extracted from the data of the plot output object (e.g. if this plot was created from `ggplotly`), otherwise directly from the plot object (if the plot was created from `plot_ly` directly).
`idVarPlot`	String with variable in the `plotly` output containing IDs.

`labelVarPlot`	String with plotly variable used to extract label to build the file name of the zip compressed file containing patient report. If not specified, the label are extracted based on the `idVarPlot` of the selected plot element.
`highlightOn`	String with event to turn on the selection (`on` parameter of `highlight`), 'plotly_click' by default.
`highlightOff`	String with event to turn off the selection (`off` parameter of `highlight`), 'plotly_doubleclick' by default.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`keyVar`	String with unique key variable, identifying unique group for which the link between the table and the plot should be done.
`keyHighlightBox`	Logical, if TRUE (FALSE by default) a selectize box is included to highlight selected element(s) of the key variable.
`labelVars`	Named character vector containing variable labels.
`verbose`	Logical, if TRUE report progress messages during execution (included in the browser 'Console').

Value

Updated `plotly` object.

Author(s)

Laure Cougnaud

`formatToHierarchicalData` *Format data to a hierarchical data, in the format as required by the plotly sunburst and treemap.*

Description

Note that new variables are created for each variable of interest (the variables are not overwritten) to avoid issues with cases where the value in the child and parent variables are the same.

Usage

```
formatToHierarchicalData(data, vars, valueVar)
```

Arguments

`data`	Data.frame with data.
`vars`	Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).
`valueVar`	String with numeric variable of `data` containing the value to display.

Value

Updated data.frame with `vars` in hierarchical format, with extra attributes (in 'metadat'):

- 'varID': String with column of output containing ID of specific element.
  This is a combination from the specified `vars`, or 'Overall' for the grand total.
- 'varParent': String with column of output containing ID of the parent element
- 'varLabel': String with column of output containing the label to display.
  This is usually the name of the child element.

Author(s)

Laure Cougnaud

`getAxisLab` *Get axis label*

Description

Get axis label

Usage

```
getAxisLab(axisVar, axisLab, labVar = NULL, data, labelVars)
```

Arguments

`axisVar`	String with column of `data` containing variable for the axis.
`axisLab`	(optional) String with label for `axisVar`
`labVar`	String with column of `data` containing variable(s) whose content should be displayed in the axis (with its labels).
`data`	Data.frame with data.
`labelVars`	Named character vector containing variable labels.

Value

String with label for the axis.

`getAxisLabs` *Set different variables for the x-axis labels*

Description

Set different variables for the x-axis labels

Usage

```
getAxisLabs(data, var, labVars)
```

Arguments

`data`	Data.frame with data.
`var`	String with variable displayed in the axis.
`labVars`	Character vector with variable(s) to be displayed as the labels of the ticks in the axis.

Value

Named character vector. The names are the position in the x-axis, the values are the new labels.

Author(s)

Laure Cougnaud

`getAxisLimPlot` *Get axis limits for a `ggplot` plot from the input dataset.*

Description

Get axis limits for a `ggplot` plot from the input dataset.

Usage

```
getAxisLimPlot(
  data,
  xVar,
  yVar,
  xLim = NULL,
  yLim = NULL,
  facetPars = NULL,
  refLineData = NULL
)
```

Arguments

`data`	Data.frame with data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLim`, `yLim`	Numeric vector of length 2 with limits for the x/y axes.
`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.
`refLineData`	Data used for the reference lines, as output of the `getDataReferenceLines` function.

Value

Data.frame with limits of the:

- x-axis: 'xmin'/'xmax'
- y-axis: 'ymin'/'ymax'

for each element of the facetting variable (if any).

`getDataReferenceLines` *Extract data for the reference lines*

Description

This function especially extracts the data if an aesthetic variable is specified in the reference line parameters.

Usage

```
getDataReferenceLines(refLinePars, data, facetPars = NULL)
```

Arguments

`refLinePars`	(optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains: • aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`. • 'label': (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.
`data`	Data.frame with data.
`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.

Value

List of data for the lines

Author(s)

Laure Cougnaud

`getDimGgplot` *Get plot dimensions*

Description

Get plot dimensions

Usage

```
getDimGgplot(gg)
```

Arguments

`gg`	`ggplot`

Value

Numeric vector with number of rows ('nrow') and columns ('ncol') of the plot

Author(s)

Laure Cougnaud

`getExtraDirs` *Get extra directory(ies) required for the clinical data review report*

Description

By default, the 'figures', 'tables' and patient profiles folders (`patientProfilePath` parameter in the general config file, if specified) in the input directory are considered.

Usage

```
getExtraDirs(inputDir = ".", configDir = file.path(inputDir, "config"))
```

Arguments

`inputDir`	String with input directory, working directory by default.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.

Value

Character vector with extra directories required by the report

Author(s)

Laure Cougnaud

`getFacetVars` *Get facetting variables from facet parameters.*

Description

Get facetting variables from facet parameters.

Usage

```
getFacetVars(facetPars = list())
```

Arguments

`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.

Value

Character vector with facetting variable

Author(s)

Laure Cougnaud

`getFctCode` *Get function code*

Description

Get function code

Usage

```
getFctCode(fct)
```

Arguments

`fct`	a R function

Value

String with function code

`getFctTypeReferenceLines` *Get the names of the `ggplot` function to use for the reference lines*

Description

Get the names of the `ggplot` function to use for the reference lines

Usage

```
getFctTypeReferenceLines(refLinePars)
```

Arguments

`refLinePars` (optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains:

- aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`.
- 'label': (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.

Value

List of type of each reference lines, among: 'vline', 'hline' and 'abline'.

Author(s)

Laure Cougnaud

`getHeightLab` *Get height of labels: title, subtitle or caption*

Description

Get height of labels: title, subtitle or caption

Usage

```
getHeightLab(lab)
```

Arguments

`lab`	String with label.

Value

Integer with height in pixels for this element.

Author(s)

Laure Cougnaud

`getHTMLToc` *Get HTML toc*

Description

Get HTML toc

Usage

```
getHTMLToc(toc)
```

Arguments

`toc`	data.frame with TOC info

Value

Character vector with HTML toc

`getIndexHTMLTitle` *Get index of the line containing the HTML title in a vector of HTML strings*

Description

Get index of the line containing the HTML title in a vector of HTML strings

Usage

```
getIndexHTMLTitle(x)
```

Arguments

`x`	Character vector with HTML

Value

Integer vector with index of the title in the vector `x`

`getInterimResFile` *Get interim res file*

Description

Get interim res file

Usage

```
getInterimResFile(intermediateDir = "./interim", mdFile)
```

Arguments

`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`mdFile`	String with path of the Markdown file

Value

String with path to the file with intermediate results.

`getJitterVar` *Add jitter to the variable of the plot, based on the different groups of a grouping variable*

Description

Add jitter to the variable of the plot, based on the different groups of a grouping variable

Usage

```
getJitterVar(data, var, byVar)
```

Arguments

`data`	Data.frame with data.
`var`	String with variable to add a jitter to.
`byVar`	String with variable containing the groups to jitter by.

Value

Numeric vector of length `nrow(data)` containing the jittered variable.

Author(s)

Laure Cougnaud

`getJsDepClinDataReview` *Get Javascript custom scripts required for specific clinical data functionalities.*

Description

Get Javascript custom scripts required for specific clinical data functionalities.

Usage

```
getJsDepClinDataReview(
  type = c("collapsibleButton", "patientProfiles"),
  dep = NULL
)
```

Arguments

`type`	(optional) Character vector with type of dependencies, either: 'collapsibleButton' or 'patientProfiles'.
`dep`	(optional) Character vector with names of Javascript dependencies By default, all dependencies are included.

Value

List of `htmlDependency`. To include this dependency in a report e.g. generated with rmarkdown, these can be passed to the: `extra_dependencies` parameter of the `output_format` specific function, e.g.: `rmarkdown::render(...,output_format = rmarkdown::html_document(extra_dependencies = dep))`

Author(s)

Laure Cougnaud

`getMdFromConfig` *Get path of the* `HTML` *file corresponding to a specific config file.*

Description

The name of the `Markdown` file is based on:

- for the general `config.yml` file: the basename of the specified `indexPath`
- for other config file (each sub-report): the name of the config file, after removal of the '`config-`' part.

Usage

```
getMdFromConfig(
  configFiles,
  indexPath = "index.Rmd",
  intermediateDir = "./interim"
)
```

Arguments

`configFiles`	Character vector with name or path of the config file(s).
`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.

Value

String with path to the HTML file

Author(s)

Laure Cougnaud

`getMdHeader` *Get Markdown header, for creation in Rmarkdown.*

Description

The depth is extracted:

1. from the `settings` if a specified depth is provided in the 'rmd_file_depth' for the current knitted file
2. `level` parameter otherwise

Usage

```
getMdHeader(title, level = 1)
```

Arguments

`title`	String with header title.
`level`	Integer of length 1 with header depth/level, 1 by default

Value

String with Markdown header, to be included in R within `cat`.

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`, `render_clinDataReviewReport()`

`getMetadata` *Read metadata file*

Description

Read the metadata file from a yaml format. This function checks for existance of the metadata file and its content. In particular, within the yaml file matches the following strings:

- `path`: Path to the data. More than one path is allowed.
- `dateTime`: Date and time, usually of the SDTM data creation. When printing the metadata in Rmd document, there is the possibility to add the date and time of the report generation. See `knit_print.clinDataReviewMetadata`.
- `datasetInfo`: General information about the data sets.

Usage

```
getMetadata(filePath, namesInfo)
```

Arguments

`filePath`	String of path to file. Currently only one file path is supported. If more than one paths are provided, a warning will be printed and the first path will be used.
`namesInfo`	Named vector to rename the final output when printed in Rmd. The renaming happens only if the metadata info are printed in Rmd and not in the console.

Details

Note that the input names do not necessary have to match the exact names. For instance, the user can also write "dataTimeMySDTMData", and the function will parse for existance of the string "dataTime".

Value

A list of:

- `summaryInfo` Information extracted from the inputs `path`, and `dateTime`.
- `datasetInfo` Information extracted from `datasetInfo`.

Examples

```
# Create temporary yaml file
tmpdir <- tempdir()
library(yaml)

tmpYamlFile <- tempfile(
    pattern = "file", tmpdir = tempdir(), fileext = ".yml"
)
listArgs <- list(
```

```
    pathSDTMs = "path/To/SDTM",
    pathSDTMReformat = "path/To/SDTMReformat",
    dateTime = "20200101",
    datasetInfo = list(
        list(
            column1 = "ex.xpt",
            column2 = "20200101"
        ),
        list(
            column1 = "sl.xpt",
            column2 = "20200101",
            column3 = "OK"
        )
    )
)
write_yaml(
    listArgs,
    file = tmpYamlFile
)

# Run metadata
# Note: the 'datasetInfo' can also contain empty elements
getMetadata(filePath = tmpYamlFile)
```

`getParamsFromConfig` *Get parameters from a config file*

Description

Please note that the information from this config file and the general config file: `config.yml` are considered.
In case parameters are defined both in the general and specific config files, the parameter from the general config file is ignored.

Usage

```
getParamsFromConfig(
  configFile,
  configDir = file.path(inputDir, "config"),
  inputDir = "."
)
```

Arguments

`configFile`	String with filename of the config file of interest in YAML format.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`inputDir`	String with input directory, working directory by default.

Value

List with parameters from the specified `configFile` and the general config file: `config.yml`. There are two specific handlers:

- parameters tagged with '[param] !r [value]' are evaluated in R, and their evaluated value is returned
- parameters tagged with '[param] !r-lazy [value]' are imported as character, and need to be further process with `forceParams` inside the report.

Parameters with YAML type 'r-lazy' are imported as character, with this additional class.

Author(s)

Laure Cougnaud

See Also

forceParams

Other clinical data reporting: `checkReportTitles`(), `forceParams`(), `getMdHeader`(), `gitbook_clinDataReview_repor`
`html_clinDataReview_report`(), `knitPrintClinDataReview`(), `postProcessReport`(), `render_clinDataReviewRepo`

`getParFctReferenceLines`

Get parameter of function used for reference lines

Description

Get parameter of function used for reference lines

Usage

```
getParFctReferenceLines(type)
```

Arguments

`type`	string with line type, either: 'hline', 'abline' or 'vline'.

Value

Character vector with parameter names of the functions

Author(s)

Laure Cougnaud

`getPathHyperlink` *Get path ('href') property from hyperlink(s).*

Description

Get path ('href') property from hyperlink(s).

Usage

```
getPathHyperlink(x)
```

Arguments

`x`	Character vector with hyperlink(s). If multiple, the hyperlinks should be separated by: ', '.

Value

Character vector of length `x` containing only the hyperlinks.

Author(s)

Laure Cougnaud

`getPathTemplate` *Get path of template clinical data report*

Description

Get path of template clinical data report

Usage

```
getPathTemplate(file, package = "clinDataReview")
```

Arguments

`file`	String with name of the template Rmd document
`package`	String, which package the template should be extracted from, by default the `clinDataReview` package.

Value

String with path to the template in the installed `clinDataReview` package

Author(s)

Laure Cougnaud

Examples

```
## Not run:
pathDivisionTemplate <- getPathTemplate("divisionTemplate.Rmd") # get path template in the package
file.copy(from = pathDivisionTemplate, to = ".") # copy to current directlory
rmarkdown::render(pathDivisionTemplate) # run file

## End(Not run)
```

`getPlotTableVars` *Extract variables displayed in the attached table, for each available plotting function of the clinDataReview package.*

Description

This function is used in each plotting function of the package to extract the variable(s) displayed in the table associated to the plot and their associated labels.
This can also be used in the template reports, e.g. to extract reference variable(s) for the comparison table functionality in the plot template report.
The following framework is used:

- if variables to be displayed in the table (`tableVars`) are not specified:
  all variables displayed in the plot are selected, based on the plot arguments.
  For example: the variables displayed in the x and y axis and for coloring are extracted for the `scatterplotClinData` plotting function.
  Label for these variable(s) are extracted from the associated parameter (e.g. `xLab` for `xVar` and so on) or the general parameter for the variable labels (`labelVars`) if not specified.
- if variables to be displayed in the table (`tableVars`) are specified:
  these variable(s) are returned.
  The associated label(s) are extracted from the associated parameter (`tableLab`) or the general parameter for the variable labels (`labelVars`) if not specified.

For the functions: `plotCountClinData`, `treemapClinData`, `sunburstClinData`: value to represent are included in the table and colored with a bar.

Usage

```
getPlotTableVars(plotFunction, plotArgs)
```

Arguments

`plotFunction`	String with name of the plotting function, be available in the `clinDataReview` package.
`plotArgs`	List with parameters passed to the plotting function.

Value

Character vector with variable to include in the table, with extra attributes (passed to `tableClinData`):

- 'tableLab': Named character vector with labels for the table variables
- 'tablePars' : extra table parameters, only included if specified as input or specified internally.

labels and the table parameters .

Author(s)

Laure Cougnaud

`getPositionAndMargins` *Get margins and positions of specific elements for a clinical data plot*

Description

The elements are positioned as following:

- on top of the plot
  1. title
  2. subtitle
  3. legend, if positioned on top of the plot
  4. facet title
- at the bottom of the plot
  1. label for the x-axis
  2. legend, if positioned on the bottom of the plot
  3. caption

Margins are computed based on the presence of these elements.
Only one line is counted for the legend, as plotly will extend the margin if necessary for the legend (for bottom legend).

Usage

```
getPositionAndMargins(
  title = NULL,
  subtitle = NULL,
  xLab = NULL,
  caption = NULL,
  facet = FALSE,
  includeLegend = TRUE,
  legendPosition = "right"
)
```

Arguments

`title`	String with title for the plot.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`xLab`	String with label for `xVar`.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`facet`	Logical, if TRUE the plot contains facets.
`includeLegend`	Logical, if TRUE (by default) a legend is available in the plot.
`legendPosition`	String with position of the legend, among: 'top'/'left'/'bottom'/'right', 'right' by default.

Value

List with:

- 'margin': List with bottom ('t') and top ('t') margins in pixels
- 'position': List with position of the following plot elements:
  - on top of the plot: subtitle and legend (if positioned at the top).
    The position is defined as the distance in pixels from the top of the plotting area to the bottom of the element (`yanchor = 'bottom'`)
  - at the bottom of the plot: caption, xLab and legend (if positioned at the bottom).
    The position is defined as the distance in pixels from the bottom of the plotting area to the top of the element (`yanchor = 'top'`)
    Especially, the legend should be positioned with anchor 'top' such as the margins are automatically expanded if the legend contains multiple rows.

Author(s)

Laure Cougnaud

`getSizePlot` *Get dimensions for a clinical data plot*

Description

This function set sensitive defaults dimensions for a plot in the package. This includes:

- setting a default width for a figure to fit in a standard clinical data review report
- increasing the figure height:
  - for facetted plot, ensuring that each facet is relatively squared
  - if a caption, subtitle, title, title for the x-axis are specified
  - if a legend is set at the bottom or the top of the plot
- increasing the figure width if a legend is set at the left or the right of the plot

Usage

```
getSizePlot(
  width = NULL,
  height = NULL,
  gg = NULL,
  nrow = 1L,
  ncol = 1L,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  xLab = NULL,
  facet = FALSE,
  includeLegend = TRUE,
  legendPosition = "right",
  y = NULL
)
```

Arguments

`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`gg`	`ggplot`
`nrow`	single-length integer specifying the number of facet rows in the plot. (default = 1) Overwritten if gg is specified.
`ncol`	single-length integer specifying the number of facet columns in the plot. (default = 1) Overwritten if gg is specified.
`title`	String with title for the plot.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`xLab`	String with label for `xVar`.
`facet`	Logical, if TRUE the plot contains facets.
`includeLegend`	Logical, if TRUE (by default) a legend is available in the plot.
`legendPosition`	String with position of the legend, among: 'top'/'left'/'bottom'/'right', 'right' by default.
`y`	Character vector or factor with elements in the y-axis, or list of such vectors. If a list is provided, the maximum height obtained across the different list elements is used.

Value

Numeric vector with width ('width') and height ('height') of the plot in pixels.

Author(s)

Laure Cougnaud

`getTocNumbering` *Get TOC numbering*

Description

Get TOC numbering

Usage

```
getTocNumbering(levels)
```

Arguments

`levels`	vector with levels of the section, in the order as available in the book.

Value

Character vector with section numbers

Author(s)

Laure Cougnaud

`getWatermark` *Get a watermark, to be included in a clinical visualization*

Description

Get a watermark, to be included in a clinical visualization

Usage

```
getWatermark(
  label = "EXPLORATORY",
  color = "lightgrey",
  fontsize = 20,
  file = NULL,
  width = 800,
  height = 500
)
```

Arguments

`label`	String with label for the watermark, 'EXPLORATORY' by default.
`color`	String with color for the watermark, 'lightgrey' by default.
`fontsize`	Integer with fontsize for the watermark, 20 by default.
`file`	(optional) String with path to a file where the watermark will be exported to.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.

Value

The `file` with the watermark (invisibly).

Author(s)

Laure Cougnaud, Kirsten van Hoorde

See Also

Other watermark helpers: `addWatermark()`

Examples

```
# export a watermark to a file
watermark <- tempfile(pattern = "watermark", fileext = ".png")
getWatermark(file = watermark)
```

`gitbook_clinDataReview_report`

Clinical data format for bookdown report.

Description

This function is only meant to set sensitive defaults for gitbook.
`gitbook` can be used instead.

Usage

```
gitbook_clinDataReview_report(
  logo = NULL,
  logoHeight = "60px",
  split_by = "section+number",
  config = list(sharing = NULL, toc = list(collapse = "section")),
  extra_dependencies = NULL,
  css = NULL,
  ...
)
```

Arguments

`logo`	String, path to the logo. No logo is printed by default.
`logoHeight`	String, indicating the logo height; 60px height by default.
`split_by`	String, how the reports should be split, (see help of the gitbook function)
`config`	List with config parameters, by default: no sharing and collapsed by section. (see help of the gitbook function)
`extra_dependencies`	NULL by default
`css`	String, path to the css.
`...`	Extra parameters passed to the gitbook function.

Value

R Markdown output format to pass to `render_book`.

Author(s)

Laure Cougnaud

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`, `render_clinDataReviewRepo`

`html_clinDataReview_report`

Clinical data format for rmarkdown report.

Description

This function only kept for back-compatibility, `html_document` can be used instead.

Usage

```
html_clinDataReview_report(extra_dependencies = NULL, ...)
```

Arguments

`extra_dependencies`	NULL by default.
`...`	Extra parameters passed to the html_document function.

Value

R Markdown output format to pass to `render`.

Author(s)

Laure Cougnaud

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`, `render_clinDataReview`

`JSONSchToRd` *Get R Documentation from a JSON schema.*

Description

Note: this function doesn't support the full JSON schema specification, currently only the functionalities required by the templates of the package are implemented.

Usage

```
JSONSchToRd(JSONSch, title = NULL)
```

Arguments

`JSONSch`	List with JSON schema, as returned by `fromJSON`.
`title`	(optional) String with title. This will combined with the JSON schema 'title' tag if this is specified. is not available.

Value

Character vector with R documentation for the specified JSON schema.

Supported JSON schema tags

- 'title' is used as Rd section header
- 'description' is included in the text
- parameters are extracted from the following 'properties' tag:
  - 'type': object type
  - 'doc': documentation for the parameter (custom JSON schema tag). This can contain any Roxygen tags, e.g.: `\link[package]{function}`.
  - 'pattern' (optional): required value for the parameter
  - 'items' (optional): JSON schema for the different elements of an 'object'
  - 'minItems'/'maxItems' (optional): minimum/maximum number of elements in an 'array'
  - 'enum' (optional): set of possible values
  - 'const' (optional): fixed value for the parameter (a.k.a 'constant')

  If a parameter is required, it should be listed in the 'required' tag of the schema (outside of the 'properties' tag).

Author(s)

Laure Cougnaud

`knitPrintClinDataReview` *Include output from clinical data, or list of such outputs in a Rmarkdown report, with an appropriate title.*

Description

Include output from clinical data, or list of such outputs in a Rmarkdown report, with an appropriate title.

Usage

```
knitPrintClinDataReview(list, sep = ".", level = 1)
```

Arguments

`list`	Named list of clinical data plots, the names are used for the section header. If several section header should be created, either: • a list of level 1 named by the different group elements, separated by `sep`, e.g. `list('group1.param1' = .., 'group1.param2' = ...)`. Such list is e.g. created with `dlply`. • a nested list, named with the different groups, e.g. created with `lapply`
`sep`	String with separator used to distinguish different levels in the labels of the list. e.g. '.' by default.
`level`	Integer with base level for section, 1 by default.

Value

No returned value, the plots are included in the report. If a element in the list are empty (NULL), these elements (and the associated sections) are not included in the report.

Author(s)

Laure Cougnaud

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `html_clinDataReview_report()`, `postProcessReport()`, `render_clinDataReviewReport()`

`knit_print.clinDataReview` *Print* `clinDataReviewTable` *object in a knitted document (e.g. Rmarkdown document).*

Description

Print `clinDataReviewTable` object in a knitted document (e.g. Rmarkdown document).

Usage

```
## S3 method for class 'clinDataReview'
knit_print(x, ...)
```

Arguments

`x`	Object of class `clinDataReview`
`...`	Extra parameters for compatibility with `knit_print`, not used currently.

Author(s)

Laure Cougnaud

`knit_print.clinDataReviewMetadata` *Print metadata file in the clinical data report*

Description

This function receives the metadata information from `getMetadata` and prints them in a format for an Rmd report. In general, any list could be called as long as it is composed by two elements:

- `summaryInfo`: an R object.
- `datasetInfo`: a data.frame or a matrix.

The first (`summaryInfo`) is printed as `kable` object and the second (`datasetInfo`) is printed as hide/show html button with the function `collapseHtmlContent`.

Usage

```
## S3 method for class 'clinDataReviewMetadata'
knit_print(x, options = list(), ...)
```

Arguments

`x`	List of two elements named `summaryInfo` and `datasetInfo`.
`options`	List of extra options to be passed as chunk options. The option `dateReportRun` sets to true prints the date and time of the report creation.
`...`	Extra arguments to be passed.

Value

Nothing. The tables are ready to be printed in Rmd.

html code to include metadata in a report

`layoutClinData` *Set layout for a clinical data plot.*

Description

Set layout for a clinical data plot.

Usage

```
layoutClinData(
  xLab = NULL,
  yLab = NULL,
  title = NULL,
  caption = NULL,
  subtitle = NULL,
  includeLegend = FALSE,
  legendPosition = "right",
  facet = FALSE,
  nrow = 1L,
  ncol = 1L,
  width,
  height,
  watermark = NULL,
  ...
)
```

Arguments

`xLab`	String with label for `xVar`.
`yLab`	String with label for `xVar`.
`title`	String with title for the plot.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.

`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`includeLegend`	Logical, if TRUE (by default) a legend is available in the plot.
`legendPosition`	String with position of the legend, among: 'top'/'left'/'bottom'/'right', 'right' by default.
`facet`	Logical (FALSE by default), does the plot contains facets?
`nrow`	single-length integer specifying the number of facet rows in the plot. (default = 1) Overwritten if gg is specified.
`ncol`	single-length integer specifying the number of facet columns in the plot. (default = 1) Overwritten if gg is specified.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`watermark`	(optional) String with path to a file containing a watermark.
`...`	Any parameters for the `layout` function. This should contain at least the plot object.

Value

The updated `plotly` object

Author(s)

Laure Cougnaud

`merge.sessionInfo` *Merge multiple session information*

Description

Merge multiple session information

Usage

```
## S3 method for class 'sessionInfo'
merge(...)
```

Arguments

`...`	objects of type `sessionInfo`

Value

`sessionInfo` with combined information

Author(s)

Laure Cougnaud

`moveSkeletonFiles` *Move skeleton files from the package to a directory*

Description

This function moves the files used to create the skeleton from the package to a specified directory.

Usage

```
moveSkeletonFiles(dir)
```

Arguments

`dir`	String, path to the directory.

Value

Nothing, the files are available in the specified directory.

`moveXpt` *Move data sets from clinUtils*

Description

Move SDTM data sets available in `clinUtils` into a specified local directory.

Usage

```
moveXpt(dir)
```

Arguments

`dir`	String, path to the directory.

Value

Nothing, the data are saved in the dedicated location.

`plotCountClinData` *Interactive plot of 'count' data*

Description

Note: the table and plot are not (yet) linked.

Usage

```
plotCountClinData(
  data,
  vars,
  varsLab = getLabelVar(vars, labelVars = labelVars),
  valueVar,
  valueLab = getLabelVar(valueVar, labelVars = labelVars),
  colorVar = NULL,
  colorLab = getLabelVar(valueVar, labelVars = labelVars),
  colorPalette = getOption("clinDataReview.colors"),
  colorRange = NULL,
  valueType = "total",
 title = paste(c(paste(valueLab, "by", paste(varsLab, collapse = " and ")), titleExtra),
    collapse = "<br>"),
  titleExtra = NULL,
  subtitle = NULL,
  caption = NULL,
  labelVars = NULL,
  width = NULL,
  height = NULL,
  pathVar = NULL,
  pathLab = getLabelVar(pathVar, labelVars = labelVars),
  hoverVars = c(vars, valueVar, colorVar),
  hoverLab = getLabelVar(hoverVars, labelVars = labelVars),
  table = FALSE,
  tableVars,
  tableLab,
  tableButton = TRUE,
  tablePars = list(),
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  verbose = FALSE,
  typePlot = c("sunburst", "treemap"),
  watermark = NULL
)
```

Arguments

`data`	Data.frame with data.

`vars`	Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).
`varsLab`	Named character vector with labels for `vars`.
`valueVar`	String with numeric variable of `data` containing the value to display.
`valueLab`	String with label for the `valueVar` variable.
`colorVar`	(optional) String with coloring variable (NULL by default). By default, the treemap is colored based by section.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`colorRange`	(optional) Numeric vector of length 2 with range for the color variable, in case it is a numeric variable.
`valueType`	String with type of values in `valueVar` (`branchvalues` of the `plot_ly`) function), among others: 'total' (default, only if sum(child) <= to parent) or 'relative'.
`title`	String with title for the plot.
`titleExtra`	String with extra title for the plot (appended after `title`).
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`labelVars`	Named character vector containing variable labels.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be: • compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.

`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`id`	String with general id for the plot: • ’id’ is used as `group` for the `SharedData` • ’button:[id]’ is used as button ID if `table` is TRUE If not specified, a random id, as ’plotClinData[X]’ is used.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.
`typePlot`	String with plot type, ’treemap’ or ’sunburst’.
`watermark`	(optional) String with path to a file containing a watermark.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the ’plot’ (`plotly` object) and ’table’ (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other visualizations of summary statistics for clinical data: `barplotClinData()`, `boxplotClinData()`, `errorbarClinData()`, `sunburstClinData()`, `treemapClinData()`

`postProcessReport` *Convert clinical data Markdown files to HTML*

Description

Convert clinical data Markdown files to HTML

Usage

```
postProcessReport(
  inputDir = ".",
  configDir = file.path(inputDir, "config"),
  indexPath = file.path(inputDir, "index.Rmd"),
  extraDirs = getExtraDirs(inputDir = inputDir, configDir = configDir),
  outputDir = "./report",
  intermediateDir = "./interim",
  mdFiles = NULL,
  nCores = 1,
  logFile = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`inputDir`	String with input directory, working directory by default.
`configDir`	String with directory with config files, by default a ’config’ folder in `inputDir`. It should contain a general ’config.yml’ file and dedicated ’config-[X].yml’ for each chapter. The order of each chapter is specified in the ’config’ slot in the general general ’config.yml’.
`indexPath`	String with path to the index file, by default ’index.Rmd’ in `inputDir`.
`extraDirs`	Character vector with extra directories required by the report, directory with external images. By default, the directories: ’figures’, ’tables’ and mentioned in the ’patientProfilePath’ parameter of the general config file are included. All these folders should be available in `inputDir`.
`outputDir`	String with output directory, (’report’ by default).
`intermediateDir`	String with intermediate directory (’interim’ by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`mdFiles`	(optional) Path to the `Markdown` files that should be converted. If specified, the specified config files in `configDir` are ignored.
`nCores`	Integer containing the number of cores used to render the report (1 by default). If more than 1, two steps of the report creation are run in parallel across chapters: • the rendering of the Rmarkdown file to Markdown • the conversion from Markdown to HTML
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.
`...`	Any parameters passed to `render`, for expert use only.

Value

String with path to the front page of the report.

Author(s)

Laure Cougnaud

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `render_clinDataReviewReport()`

`print.clinDataReview` *Print a* `clinDataReview` *object in the console*

Description

Print a `clinDataReview` object in the console

Usage

```
## S3 method for class 'clinDataReview'
print(x, ...)
```

Arguments

`x`	Object of class `clinDataReview`
`...`	Extra parameters for compatibility with `print`, not used currently.

Value

No returned value, the object is printed into the console.

`processData` *Process a dataset.*

Description

This function is intended to automate all data processing steps for use in the 'clinDataReview' reports using config files.

Usage

```
processData(data, processing, labelVars = NULL, ...)
```

Arguments

`data`	Data.frame with data.
`processing`	List with processing steps for the data. Each element in the list should be a named list containing the parameters for the specific processing function. The name specifies the processing step, among: • 'annotate' for `annotateData` (`annotations` parameter) • 'filter' for `filterData` (`filters` parameter) • 'transform' for `transformData` (`transformations` parameter) Multiple steps of each kind can be specified after each other (e.g. 1: filter, 2: transform, 3: filter, ...). If a filter step is specified as a list of multiple filters, the filters are run **independently of each other on the entire dataset** (see the documentation of `filters` in `filterData`). If filters should be run **sequentially**, i.e. filter from step 2 should be applied on the filtered dataset from step 1, **separated filtering steps** should be specified, e.g. `list(filter = list(var = "ANL01FL", value = "Y"),filter = list(var = "PARAM", value = "QTCF"))`
`labelVars`	Named character vector containing variable labels.
`...`	Any parameters passed to all processing functions (if this parameter is available). If specified, these parameters shouldn't be specified also in `processing`.

Value

Data.frame with processed `data`, with extra attribute: `labelVars`.

Author(s)

Laure Cougnaud

Examples

```
library(clinUtils)

data(dataADaMCDISCP01)

dataLB <- dataADaMCDISCP01$ADLBC

# filter and annotate data
processData(
  data = dataLB,
  processing = list(
    list(filter = list(var = "ANL01FL", value = "Y")),
   list(annotate = list(vars = "ANRIND", varFct = 'factor(ANRIND, levels = c("L", "N", "H"))'))
  )
)

## multiple filtering steps:

# If these are specified in the same 'filter' step condition, these are considered independently,
# and the selected records combined with an 'AND' operator.
# Example: consider only records:
# - with analysis flag AND
# - from subject with high/low measurement (for all records) for each parameter
processData(
  data = dataLB,
  processing = list(
    list(filter = list(
      list(var = "ANL01FL", value = "Y"),
      list(var = "ANRIND", value = c("L", "H"),
           postFct = any, varsBy = c("USUBJID", "PARAM"))
    )
    )
  )
)

# a custom operator to combine the selected records can be specified
# Example: consider only records:
# - with analysis flag OR
# - from subject with high/low measurement (for all records) for each parameter
processData(
  data = dataLB,
  processing = list(
    list(filter = list(
      list(var = "ANL01FL", value = "Y"),
      "|",
      list(var = "ANRIND", value = c("L", "H"),
           postFct = any, varsBy = c("USUBJID", "PARAM"))
    )
    )
  )
)
```

```
# If the filtering conditions are specified in different filtering steps, these are
# considered sequentially.
# Example:
# 1) consider only analysis records and
# 2) from these records, consider only subject with high/low measurement for
# each parameter
processData(
  data = dataLB,
  processing = list(
    list(filter = list(var = "ANL01FL", value = "Y")),
    list(filter = list(var = "ANRIND", value = c("L", "H"),
      postFct = any, varsBy = c("USUBJID", "PARAM")))
  )
)
# Note for this particular
```

`renamePathDateInfoMetadata` *Rename variable names of metadata info*

Description

Rename variable names referring to the paths and the date.

Usage

```
renamePathDateInfoMetadata(summaryInfo, namesInfo)
```

Arguments

`summaryInfo`	A matrix, see output from `getMetadata`.
`namesInfo`	Named vector to rename the final output.

Value

A matrix, same as input `summaryInfo` with renamed variable names.

`renderChapter` *Render one chapter of a clinical report, based on a configuration file*

Description

Render one chapter of a clinical report, based on a configuration file

Usage

```
renderChapter(
  configFile,
 configGeneralParams = getParamsFromConfig(configFile = "config.yml", configDir =
    configDir, inputDir = inputDir),
  configDir = file.path(inputDir, "config"),
  indexPath = file.path(inputDir, "index.Rmd"),
  inputDir = ".",
  intermediateDir = "./interim",
  logFile = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`configFile`	String with filename of the config file of interest in YAML format.
`configGeneralParams`	List with parameters from the general config file
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`inputDir`	String with input directory, working directory by default.
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.
`...`	options passed to `renderFile`

Value

No output file, the Markdown report for the chapter and the `knit_meta` object is available in the `intermediateDir` directory.
If the input parameters are not correctly extracted, NULL is returned.

`renderFile` *Render a rmarkdown file, possibly in a new R session*

Description

This has the possibility to save output in a log file, and saving also session information.

Usage

```
renderFile(input, encoding = "UTF-8", params = NULL, logFile = NULL, ...)
```

Arguments

`input`	Input file to be rendered.
`encoding`	String with encoding, 'UTF-8' by default.
`params`	List with input parameters for this document. These parameters should be accessed in the Rmd document via `params$...`. These parameters will be saved to a RDS file and imported during the report rendering.
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`...`	Any extra parameters passed to `render`, for expert use only.

Details

Note: this function is inspired from `xfun::Rscript_call`

Value

Output of the function with additional attribute: 'sessionInfo' containing the details of the session information. If the report fails, an error message is returned.

Author(s)

Laure Cougnaud

`render_clinDataReviewReport`

Render a clinical data review report.

Description

Render a clinical data review report.

Usage

```
render_clinDataReviewReport(
  configFiles = NULL,
  configDir = file.path(inputDir, "config"),
  logFile = NULL,
  indexPath = file.path(inputDir, "index.Rmd"),
  inputDir = ".",
  outputDir = "./report",
  intermediateDir = "./interim",
  extraDirs = getExtraDirs(inputDir = inputDir, configDir = configDir),
  quiet = FALSE,
  verbose = TRUE,
  nCores = 1
)
```

Arguments

`configFiles`	(optional) Character vector with specific config files to be converted from Rmarkdown to Markdown. If • not specified (by default): all config files specified in the general 'config.yml' will be run (Rmd -> md) • specified (**expert use only**): only the specified files will be run (Rmd -> md). Other config files mentioned in the general 'config.yml' file won't be rerun, so the associated 'md' file should be already available in the `intermediateDir` folder.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`logFile`	(optional) String with path to a log file, where output (also error/messages/warnings) should be stored. If specified, the entire output is re-directed to this file.
`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`inputDir`	String with input directory, working directory by default.
`outputDir`	String with output directory, ('report' by default).

`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`extraDirs`	Character vector with extra directories required by the report, directory with external images. By default, the directories: 'figures', 'tables' and mentioned in the 'patientProfilePath' parameter of the general config file are included. All these folders should be available in `inputDir`.
`quiet`	Logical, if TRUE (FALSE by default) messages during the execution of each report are not displayed in the console (see `render`).
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.
`nCores`	Integer containing the number of cores used to render the report (1 by default). If more than 1, two steps of the report creation are run in parallel across chapters: • the rendering of the Rmarkdown file to Markdown • the conversion from Markdown to HTML

Value

String with path to the front page of the report.

Process

This function is based on the render_book function, with the extra functionalities:

- specification of chapter-specific input parameters, specified in YAML configuration files
- (optional) creation of each chapter in parallel if `nCores` > 1. In that case, all chapters are run in parallel, excepted the chapter(s) run internally in parallel (config file with `parallel` set to 'TRUE').
- (optional) split of each chapter into html file specific for each chapter, by specifying the `split_by` parameter in the chapter-specific config file

This consists of:

1. importing the general config file ('config'.yml) to identify each report of interest ('config' tag)
2. for each report of interest:
    - loading the report specific parameters from the associated 'config' file (see the `getParamsFromConfig` function)
    - if the template should be extracted from a specified package (`templatePackage` tag), this template is copied to the current directory. Please note that if a file with same name is available in the working directory, this file will be overwritten.
    - running the report ('template' tag) with the associated parameters in a **new R session for reproducibility**, to obtain the associated Markdown file.
      This step is parallelized across the different config files, if the `nCores` parameter is specified.
3. checking if the associated `Markdown` and `rds` file (list of Js dependencies) are available in `intermediateDir`

4. split each chapter into separated Markdown documents, based on the `split_by` parameter (specified at the report or config level)
5. conversion of each Markdown document to an HTML document.
   This step is parallelized across the different Markdown documents, if the `nCores` parameter is specified.
6. build the book:
   (a) creation of a common TOC for the book
   (b) inclusion of the TOC in each Markdown file
   (c) update of the section number in each chapter
   (d) inclusion of the section number in each HTML file name

If the execution of a specific report fails with error, a warning message is triggered. A report containing only the specified title is created, to ensure output consistency (especially html file numbering) in case the report succeeds.

Available template report

see `?`‘clinDataReview-templates‘`` for a list of clinical data template report available in the package.

Extension to chapter-specific split

The bookdown ’split_by’ parameter is extended, to support:

- chapter-specific split, specified in the configuration file of the specific chapter, via the `split_by` parameter
- specification as a number (if specified within a config file), e.g. ’0’ for no split, 1’ for chapter, ’2’ for section, ’3’ for subsection, ...
- split at section level higher than 2 (until 7) (if specified within a config file)

Author(s)

Laure Cougnaud

See Also

Other clinical data reporting: `checkReportTitles()`, `forceParams()`, `getMdHeader()`, `getParamsFromConfig()`, `gitbook_clinDataReview_report()`, `html_clinDataReview_report()`, `knitPrintClinDataReview()`, `postProcessReport()`

`scatterplotClinData` *Scatterplot of variables of interest for clinical data visualization.*

Description

The parameters for this visualization are based on `ggplot2` (aesthetic, scale, ...), parameter specification, unlike the other visualizations of the package.

Usage

```
scatterplotClinData(
  data,
  xVar,
  yVar,
  xLab = getLabelVar(var = xVar, labelVars = labelVars),
  xLabVar = NULL,
  yLab = getLabelVar(var = yVar, labelVars = labelVars),
  yLabVar = NULL,
  aesPointVar = list(),
  pointPars = list(),
  aesLineVar = list(),
  linePars = list(),
  lineInclude = length(aesLineVar) > 0,
  aesSmoothVar = list(),
  smoothPars = list(),
  smoothInclude = length(c(aesSmoothVar, smoothPars)) > 0,
  aesLab,
  xTrans = "identity",
  yTrans = "identity",
  xPars = list(),
  yPars = list(),
  xLabVars = NULL,
  yLim = NULL,
  xLim = NULL,
  yLimExpandData = TRUE,
  xLimExpandData = TRUE,
  title = paste(c(paste(yLab, "vs", xLab), titleExtra), collapse = "<br>"),
  titleExtra = NULL,
  caption = NULL,
  subtitle = NULL,
  facetPars = list(),
  facetType = c("wrap", "grid"),
  scalePars = list(),
  themePars = list(legend.position = "bottom"),
  refLinePars = NULL,
  labelVars = NULL,
  width = NULL,
```

```
  height = NULL,
  hoverVars,
  hoverLab,
  idVar = "USUBJID",
  idLab = getLabelVar(idVar, labelVars = labelVars),
  idHighlightBox = FALSE,
  pathVar = NULL,
  pathExpand = FALSE,
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  selectVars = NULL,
  selectLab = getLabelVar(selectVars, labelVars = labelVars),
  table = FALSE,
  tableVars,
  tableLab,
  tableButton = TRUE,
  tablePars = list(),
  watermark = NULL,
  verbose = FALSE
)
```

Arguments

`data`	Data.frame with input data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLab`	String with label for `xVar`.
`xLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the x-axis.
`yLab`	String with label for `xVar`.
`yLabVar`	(optional) Character vector with column(s) of `data` containing variable(s) to display in the label of the y-axis.
`aesPointVar`	List with specification of aesthetic variable(s), for the point, passed to the `mapping` parameter of `geom_point`, e.g. `list(color = "TRTP")`. Please note by default symbols with fill and color are used. Color is used for the outside of the points, fill for the inside and the hover. Usually, you might want to specify both filling and coloring.
`pointPars`	List with parameters other than aesthetic variables to pass to `geom_point`, defaults to empty list.
`aesLineVar`	List with specification of aesthetic variable(s), for the line, passed to the `mapping` parameter of `geom_line`, e.g. `list(group = "USUBJID")`.
`linePars`	List with parameters other than aesthetic variables to pass to `geom_line`, defaults to empty list.
`lineInclude`	Logical, if TRUE (by default if `aesLineVar` is specified) include a scatterplot.
`aesSmoothVar`	List with specification of aesthetic variable(s), for the smoothing layer, passed to the `mapping` parameter of `geom_smooth` defaults to empty list.

`smoothPars`	List with parameters other than aesthetic variables to pass to `geom_smooth`, defaults to empty list. Note this parameter overwrites other parameters set by `aesSmoothVar`.
`smoothInclude`	Logical, if TRUE (by default if one of `aesSmoothVar` or `smoothPars` is non-empty)
`aesLab`	Named character vector with labels for each aesthetic variable.
`xTrans`, `yTrans`	Transformation for the x/y- variables, passed to the `trans` parameter of `scale_x_continuous`/ `scale_y_continuous`.
`xPars`, `yPars`	List with extra parameters for x/y axis, passed to the `scale_x_continuous`/ `scale_y_continuous` functions, besides `trans` and `limits`.
`xLabVars`	Character vector with variable(s) to be displayed as the labels of the ticks in the x-axis. By default, `xVar` is displayed. If specified, this overwrites any `labels` specified via `xPars`. In case the variable(s) contain different elements by `xVar` or between facets, they are combined and displayed below each other.
`xLim`, `yLim`	Numeric vector of length 2 with limits for the x/y axes.
`xLimExpandData`, `yLimExpandData`	Logical (TRUE by default), should the limits specified via `xLim`/`yLim` be expanded to include any data points outside of these limits? Please note that the same limits are set for all facets.
`title`	String with title for the plot.
`titleExtra`	String with extra title for the plot (appended after `title`).
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`facetPars`	List with facetting parameters, passed to the facetting function.
`facetType`	String with facetting type, either: • 'wrap': `facet_wrap` • 'grid': `facet_grid`
`scalePars`	List with parameters to customize scales. Each sublist should contains a set of parameters passed to the `scale_discrete_manual` function. If palette(s) are not specified, default palettes are used (see getColorPalette, getShapePalette, getLinetypePalette)
`themePars`	List with general theme parameters (see `theme`).
`refLinePars`	(optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains: • aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`.

	• ’label’: (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.
`labelVars`	Named character vector containing variable labels.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`hoverVars`	Character vector with variables to be displayed in the hover, by default `xVar`, `yVar` and any aesthetic variables.
`hoverLab`	Named character vector with labels for `hoverVars`.
`idVar`	String with variable containing subject ID.
`idLab`	String with label for `idVar`.
`idHighlightBox`	Logical, if TRUE (FALSE by default) a selectize box is included to highlight selected element(s) of the ID variable (`idVar`).
`pathVar`	String with variable of `data` containing path to a subject-specific report. The report info should be unique for each element of `idVar`. The report will be: • opened in a different window in the browser if the user clicks on the ’p’ (a.k.a ’profile’) key when hovering on a point of the plot • opened in the browser via hyperlink in the table
`pathExpand`	Logical, if FALSE (by default) the path to subject-report is included in an hyperlink in the table, otherwise a collapsed row is created. This should be set to TRUE only if multiple paths are included for each row in `pathVar` (e.g. in case of summary table).
`id`	String with general id for the plot: • ’id’ is used as `group` for the `SharedData` • ’button:[id]’ is used as button ID if `table` is TRUE If not specified, a random id, as ’plotClinData[X]’ is used.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. The plot and table are linked when included in a Rmarkdown document: when clicking on an plot element, only the corresponding records are retained in the associated table; when some records are selected in the table, they are highlighted in the associated table.
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`watermark`	(optional) String with path to a file containing a watermark.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other Clinical data visualization of individual profiles.: `timeProfileIntervalPlot()`

Examples

```
library(clinUtils)

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

dataLB <- dataADaMCDISCP01$ADLBC
dataDM <- dataADaMCDISCP01$ADSL
dataLB <- annotateData(dataLB, annotations = list(data = dataDM))
# subset of the data for the example
dataLB <- subset(dataLB, VISIT %in% c("SCREENING 1", "WEEK 2", "WEEK 8"))

## time profile

dataPlot <- subset(dataLB, PARAMCD == "ALT")

# with relative day
scatterplotClinData(
data = dataPlot,
xVar = "ADY",
yVar = "LBSTRESN",
aesPointVar = list(color = "TRTP", fill = "TRTP"),
aesLineVar = list(group = "USUBJID", color = "TRTP"),
labelVars = labelVars
)

# with actual visit
dataPlot$AVISIT <- with(dataPlot, reorder(trimws(AVISIT), AVISITN))
scatterplotClinData(
data = dataPlot,
xVar = "AVISIT",
yVar = "LBSTRESN",
aesPointVar = list(color = "TRTP", fill = "TRTP"),
aesLineVar = list(group = "USUBJID", color = "TRTP"),
labelVars = labelVars
)
```

```
## Not run:

# add number of subjects below each visit

if (requireNamespace("inTextSummaryTable", quietly = TRUE)) {

# compute number of subjects by visit
summaryTable <- inTextSummaryTable::computeSummaryStatisticsTable(
dataPlot,
rowVar = "AVISIT",
stats = "n"
)
# add it in the data
dataPlot <- merge(dataPlot, summaryTable[, c("AVISIT", "n")], all.x = TRUE)
dataPlot$n <- paste0("N=", dataPlot$n)

scatterplotClinData(
data = dataPlot,
xVar = "AVISIT", xLabVars = c("AVISIT", "n"),
yVar = "LBSTRESN",
aesPointVar = list(color = "TRTP", fill = "TRTP"),
aesLineVar = list(group = "USUBJID", color = "TRTP"),
labelVars = labelVars
)

}


## End(Not run)

## pairwise comparison plot of two parameters of interest:

# format data long -> wide format (one column per lab param)
dataPlot <- subset(dataLB, PARAMCD %in% c("ALT", "AST"))
dataPlot <- stats::aggregate(
LBSTRESN ~ USUBJID + VISIT + VISITNUM + PARAMCD,
data = dataPlot,
FUN = mean
)
dataPlotWide <- stats::reshape(
data = dataPlot,
timevar = "PARAMCD", idvar = c("USUBJID", "VISIT", "VISITNUM"),
direction = "wide"
)
colnames(dataPlotWide) <- sub("^LBSTRESN.", "", colnames(dataPlotWide))
# scatterplot per visit
scatterplotClinData(
data = dataPlotWide,
xVar = "ALT", yVar = "AST",
aesPointVar = list(color = "USUBJID", fill = "USUBJID"),
```

```
themePars = list(legend.position = "none"),
facetPars = list(facets = "VISIT"),
labelVars = labelVars,
subtitle = "Visualization is split by visit",
caption = "Points are colored by subject ID"
)

## Not run:

# scatterplot with all visits, link subjects
xLab <- getLabelParamcd(paramcd = "ALT", data = dataLB,
paramcdVar = "PARAMCD", paramVar = "PARAM")
yLab <- getLabelParamcd(paramcd = "AST", data = dataLB,
paramcdVar = "PARAMCD", paramVar = "PARAM")
scatterplotClinData(
data = dataPlotWide,
xVar = "ALT", yVar = "AST",
xLab = xLab,
yLab = yLab,
aesPointVar = list(color = "VISIT", fill = "VISIT"),
aesLineVar = list(group = "USUBJID"),
labelVars = labelVars
)

# scatterplot of different visits versus baseline

# add baseline as extra column:
dataPlot <- subset(dataLB, PARAMCD == "ALT")
dataPlotBL <- subset(dataPlot, VISIT == "SCREENING 1")
dataPlotBL <- dataPlotBL[with(dataPlotBL, order(USUBJID, -ADY)), ]
dataPlotBL <- dataPlotBL[!duplicated(dataPlotBL$USUBJID), ]
dataPlot$LBSTRESNBL <- dataPlot[match(dataPlot$USUBJID, dataPlotBL$USUBJID), "LBSTRESN"]

# sort visits:
dataPlot$VISIT <- with(dataPlot, reorder(VISIT, VISITNUM))

xLab <- paste(labelVars["LBSTRESN"], "for last screening visit")
yLab <- paste(labelVars["LBSTRESN"], "at visit X")
paramLab <- getLabelParamcd(paramcd = "ALT", data = dataLB,
paramcdVar = "PARAMCD", paramVar = "PARAM")
scatterplotClinData(
data = dataPlot,
xVar = "LBSTRESNBL", xLab = xLab,
yVar = "LBSTRESN", yLab = yLab,
aesPointVar = list(color = "USUBJID", fill = "USUBJID"),
aesLineVar = list(group = "USUBJID", color = "USUBJID"),
hoverVars = c("USUBJID", "VISIT", "ADY", "LBSTRESN"),
labelVars = labelVars,
facetPars = list(facets = "VISIT"),
themePars = list(legend.position = "none"),
title = paste("Comparison of actual value of",
paramLab,
"at each visit versus baseline"
```

```
),
refLinePars = list(
list(slope = 1, intercept = 0, linetype = 1, color = "black",
label = FALSE),
list(yintercept = "A1LO", linetype = 2, color = "blue"),
list(yintercept = "A1HI", linetype = 2, color = "purple",
label = "Reference Range Upper Limit")
)
)


## scatterplot with smoothing layer

data <- data.frame(
  subj = c(rep('subj1', 20), rep('subj2', 20)),
  time = rep( 1:20 , 2 ),
  response =  c(1:20, 50:31) + runif(min =-3, max = +3, 40),
  treat =  rep(c('trA', 'trB'), 20),
  stringsAsFactors = FALSE
)

# smoothing per subject
smoothPlot <- scatterplotClinData(
  data = data,
  xVar = "time", yVar = "response",
  aesPointVar = list(color = "treat"),
  aesLineVar = list(group = 'subj'),
  linePars = list(linetype='dotted'),
  aesSmoothVar = list(color='subj', group='subj'),
  smoothPars =  list(alpha=0.5, size=0.3 , se=TRUE, color = 'black')
)
smoothPlot


# plot smoothing over subjects
smoothPlot <- scatterplotClinData(
  data = data,
  xVar = "time", yVar = "response",
  aesPointVar = list(color = "treat"),
  aesLineVar = list(group = 'subj'),
  linePars = list(linetype='dotted'),
  aesSmoothVar = list(),
  smoothPars =  list(alpha=0.5, size=0.3 , se=TRUE, color = 'black')
)
smoothPlot


## End(Not run)

# add a selection box
if(interactive()){
  dataPlot <- subset(dataLB, PARAMCD == "ALT")
  dataPlot$TRTA <- with(dataPlot, reorder(TRTA, TRTAN))
```

```
    scatterplotClinData(
      data = dataPlot,
      xVar = "ADY",
      yVar = "LBSTRESN",
      aesPointVar = list(fill = "TRTA", color = "TRTA"),
      aesLineVar = list(group = "USUBJID", color = "TRTA"),
      selectVars = "TRTA",
      labelVars = labelVars
    )
  }
```

`setFacetLayoutWrap` *Set facetting layout for 'wrap' facetting.*

Description

By default, the number of columns is 2.

Usage

```
setFacetLayoutWrap(data, facetPars = list())
```

Arguments

`data`	Data.frame with data.
`facetPars`	List with facetting parameters, passed to the facetting function. Variables should be specified as character or formula. For 'wrap' facetting (`facetType` is 'wrap'), if the layout is not specified via `nrow`/`ncol`, 2 columns are used by default.

Value

Updated `facetPars`.

Author(s)

Laure Cougnaud

`setPaletteStaticScatterplotClinData` *Get standard palette for the* `staticScatterplotClinData` *function.*

Description

Get standard palette for the `staticScatterplotClinData` function.

Usage

```
setPaletteStaticScatterplotClinData(data, var, aes, scalePars, geomAes, ...)
```

Arguments

`data`	Data.frame with data for the plot.
`var`	Character vector with variable(s) to consider. If multiple, currently only the first one is considered.
`aes`	String with aesthetic, either: 'color', 'shape' or 'linetype'.
`scalePars`	List with parameters to customize scales. Each sublist should contains a set of parameters passed to the `scale_discrete_manual` function. If palette(s) are not specified, default palettes are used (see getColorPalette, getShapePalette, getLinetypePalette)
`geomAes`	List with aesthetic for each geom.
`...`	Any extra parameters than `x` and `n` for the default palette fcts.

Value

List with: `scalePars` and `geomAes`, each of those potentially updated with default palette(s).

Author(s)

Laure Cougnaud

`splitChapter` *Split a chapter based on the 'split_by' parameter.*

Description

Split a chapter based on the 'split_by' parameter.

Usage

```
splitChapter(
  configFile = NULL,
  configDir = "./config",
  mdFile = NULL,
  indexPath = "index.Rmd",
  intermediateDir = "./interim",
  outputDir = "./report",
  verbose = TRUE
)
```

Arguments

`configFile`	String with filename of the config file of interest in YAML format.
`configDir`	String with directory with config files, by default a 'config' folder in `inputDir`. It should contain a general 'config.yml' file and dedicated 'config-[X].yml' for each chapter. The order of each chapter is specified in the 'config' slot in the general general 'config.yml'.
`mdFile`	(optional) Path to the Markdown file containing the chapter. If not specified, the Markdown file corresponding to the specified `configFile` parameter is used.
`indexPath`	String with path to the index file, by default 'index.Rmd' in `inputDir`.
`intermediateDir`	String with intermediate directory ('interim' by default), where markdown files and rds file specifying Js libraries (with `knit_meta`) for each sub report are stored.
`outputDir`	String with output directory, ('report' by default).
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed during the report execution.

Value

No return value, the Markdown files are split as specified.

Extension to chapter-specific split

The bookdown 'split_by' parameter is extended, to support:

- chapter-specific split, specified in the configuration file of the specific chapter, via the `split_by` parameter
- specification as a number (if specified within a config file), e.g. '0' for no split, 1' for chapter, '2' for section, '3' for subsection, ...
- split at section level higher than 2 (until 7) (if specified within a config file)

Author(s)

Laure Cougnaud

`staticScatterplotClinData`

Scatterplot of variables of interest for clinical data visualization

---

Description

Scatterplot of variables of interest for clinical data visualization

Usage

```
staticScatterplotClinData(
  data,
  xVar,
  yVar,
  xLab = getLabelVar(xVar, labelVars = labelVars),
  yLab = getLabelVar(yVar, labelVars = labelVars),
  aesPointVar = list(),
  pointPars = list(),
  aesLineVar = list(),
  linePars = list(),
  lineInclude = length(c(aesLineVar, linePars)) > 0,
  aesSmoothVar = list(),
  smoothPars = list(),
  smoothInclude = length(c(aesSmoothVar, smoothPars)) > 0,
  aesLab,
  xTrans = "identity",
  yTrans = "identity",
  xPars = list(),
  yPars = list(),
  xLabVars = NULL,
  yLim = NULL,
  xLim = NULL,
  yLimExpandData = TRUE,
  xLimExpandData = TRUE,
  title = paste(c(paste(yLab, "vs", xLab), titleExtra), collapse = "<br>"),
  titleExtra = NULL,
  facetPars = list(),
  facetType = c("wrap", "grid"),
  scalePars = list(),
  themePars = list(legend.position = "bottom"),
  refLinePars = NULL,
  labelVars = NULL,
  hoverVars = NULL,
  geomType = c("point", "col")
)
```

Arguments

`data`	Data.frame with input data.
`xVar`	String with column of `data` containing x-variable.
`yVar`	String with column of `data` containing y-variable.
`xLab`	String with label for `xVar`.
`yLab`	String with label for `xVar`.
`aesPointVar`	List with specification of aesthetic variable(s), for the point, passed to the `mapping` parameter of `geom_point`, e.g. `list(color = "TRTP")`. Please note by default symbols with fill and color are used. Color is used for the outside of the points, fill for the inside and the hover. Usually, you might want to specify both filling and coloring.
`pointPars`	List with parameters other than aesthetic variables to pass to `geom_point`, defaults to empty list.
`aesLineVar`	List with specification of aesthetic variable(s), for the line, passed to the `mapping` parameter of `geom_line`, e.g. `list(group = "USUBJID")`.
`linePars`	List with parameters other than aesthetic variables to pass to `geom_line`, defaults to empty list.
`lineInclude`	Logical, if TRUE (by default if `aesLineVar` is specified) include a scatterplot.
`aesSmoothVar`	List with specification of aesthetic variable(s), for the smoothing layer, passed to the `mapping` parameter of `geom_smooth` defaults to empty list.
`smoothPars`	List with parameters other than aesthetic variables to pass to `geom_smooth`, defaults to empty list. Note this parameter overwrites other parameters set by `aesSmoothVar`.
`smoothInclude`	Logical, if TRUE (by default if one of `aesSmoothVar` or `smoothPars` is non-empty)
`aesLab`	Named character vector with labels for each aesthetic variable.
`xTrans`, `yTrans`	Transformation for the x/y- variables, passed to the `trans` parameter of `scale_x_continuous`/ `scale_y_continuous`.
`xPars`, `yPars`	List with extra parameters for x/y axis, passed to the `scale_x_continuous`/ `scale_y_continuous` functions, besides `trans` and `limits`.
`xLabVars`	Character vector with variable(s) to be displayed as the labels of the ticks in the x-axis. By default, `xVar` is displayed. If specified, this overwrites any `labels` specified via `xPars`. In case the variable(s) contain different elements by `xVar` or between facets, they are combined and displayed below each other.
`xLim`, `yLim`	Numeric vector of length 2 with limits for the x/y axes.
`xLimExpandData`, `yLimExpandData`	Logical (TRUE by default), should the limits specified via `xLim`/`yLim` be expanded to include any data points outside of these limits? Please note that the same limits are set for all facets.
`title`	String with title for the plot.

`titleExtra`	String with extra title for the plot (appended after `title`).
`facetPars`	List with facetting parameters, passed to the facetting function.
`facetType`	String with facetting type, either: • ’wrap’: `facet_wrap` • ’grid’: `facet_grid`
`scalePars`	List with parameters to customize scales. Each sublist should contains a set of parameters passed to the `scale_discrete_manual` function. If palette(s) are not specified, default palettes are used (see getColorPalette, getShapePalette, getLinetypePalette)
`themePars`	List with general theme parameters (see `theme`).
`refLinePars`	(optional) Nested list, with parameters for each reference line(s). Each sublist (a.k.a reference line) contains: • aesthetic value(s) or variable(s) for the lines (in this case column names of `data`) for reference lines. The line position is controlled by the aesthetics supported in `geom_vline`, `geom_hline` and `geom_abline`. • ’label’: (optional) Logical specifying if the line should be annotated (`FALSE` to not annotate the line) or string with annotation label. By default, the value of the position of the horizontal/vertical line or the equation of the diagonal line is displayed.
`labelVars`	Named character vector containing variable labels.
`hoverVars`	Character vector with variables to be displayed in the hover, by default `xVar`, `yVar` and any aesthetic variables.
`geomType`	String with type of the geom used, either: • ’point’: scatterplot with `geom_point` is created • ’col’: barplot with `geom_col` is created

Value

`ggplot` object

Author(s)

Laure Cougnaud, Adriaan Blommaert

`sunburstClinData` *Sunburst interactive plot.*

Description

Note: the table and plot are not (yet) linked.

Usage

```
sunburstClinData(...)
```

Arguments

`...` Arguments passed on to `plotCountClinData`

`colorVar` (optional) String with coloring variable (NULL by default). By default, the treemap is colored based by section.

`colorRange` (optional) Numeric vector of length 2 with range for the color variable, in case it is a numeric variable.

`vars` Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).

`varsLab` Named character vector with labels for `vars`.

`valueVar` String with numeric variable of `data` containing the value to display.

`valueLab` String with label for the `valueVar` variable.

`valueType` String with type of values in `valueVar` (`branchvalues` of the `plot_ly`) function), among others: 'total' (default, only if sum(child) <= to parent) or 'relative'.

`pathVar` String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as:

```
<a href="./path-to-report">label</a>
```

.

If multiple, they should be separated by: ', '.

The report(s) will be:

- compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot
- included in a collapsible row, and clickable with hyperlinks in the table

`pathLab` String with label for `pathVar`, included in the collapsible row in the table.

`table` Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)

`data` Data.frame with data.

`verbose` Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

`width` Numeric, width of the plot in pixels, 800 by default.

`height` Numeric, height of the plot in pixels, 500 by default.

`hoverVars` Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.

`hoverLab` Named character vector with labels for `hoverVars`.

`labelVars` Named character vector containing variable labels.

`id` String with general id for the plot:

- 'id' is used as `group` for the `SharedData`
- 'button:[id]' is used as button ID if `table` is TRUE

If not specified, a random id, as 'plotClinData[X]' is used.

`title` String with title for the plot.

`titleExtra` String with extra title for the plot (appended after `title`).

`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`watermark`	(optional) String with path to a file containing a watermark.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tablePars`	List with parameters passed to the `getClinDT` function.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other visualizations of summary statistics for clinical data: `barplotClinData()`, `boxplotClinData()`, `errorbarClinData()`, `plotCountClinData()`, `treemapClinData()`

Examples

```
library(clinUtils)

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

dataAE <- dataADaMCDISCP01$ADAE
dataDM <- dataADaMCDISCP01$ADSL

## example of basic sunburst:

# sunburst takes as input table with counts
if (requireNamespace("inTextSummaryTable", quietly = TRUE)) {

# total counts: Safety Analysis Set (patients with start date for the first treatment)
```

```
dataTotal <- subset(dataDM, RFSTDTC != "")

# compute adverse event table
tableAE <- inTextSummaryTable::getSummaryStatisticsTable(

data = dataAE,
rowVar = c("AESOC", "AEDECOD"),
dataTotal = dataTotal,
rowOrder = "total",
labelVars = labelVars,
stats = inTextSummaryTable::getStats("count"),

# plotly treemap requires records (rows) for each group
rowVarTotalInclude = "AEDECOD",
outputType = "data.frame-base"

)

dataSunburst <- tableAE

dataSunburst$n <- as.numeric(dataSunburst$n)

# create plot
sunburstClinData(
data = dataSunburst,
vars = c("AESOC", "AEDECOD"),
valueVar = "n",
    valueLab = "Number of patients with adverse events"
)

## example where sum(counts) of child = counts of parent

# counts of patients per arm/site
tableDM <- inTextSummaryTable::getSummaryStatisticsTable(
data = dataDM,
rowVar = c("ARM", "SITEID"),
labelVars = labelVars,
# plotly treemap requires records (rows) for each group
rowVarTotalInclude = "SITEID",
rowTotalInclude = TRUE,
outputType = "data.frame-base"
)
tableDM$statN <- as.numeric(tableDM$statN)

# create the plot
sunburstClinData(
data = tableDM,
vars = c("ARM", "SITEID"),
valueVar = "statN", valueLab = "Counts of patients",
valueType = "total",
caption = "The sectors are colored by category.",
subtitle = "Group: treatment and site"
)
```

```
}
```

`tableClinData` *Create a 'clinical data table', associated to a plot.*

Description

Interactive table is created, with the possibility to have clickeable link to patient-specific report, and included within a button.

Usage

```
tableClinData(
  data,
  idVar = "USUBJID",
  idLab = getLabelVar(idVar, labelVars = labelVars),
  keyVar = NULL,
  keyLab = getLabelVar(keyVar, labelVars = labelVars),
  pathVar = NULL,
  pathLab = getLabelVar(pathVar, labelVars = labelVars),
  pathExpand = FALSE,
  tableVars = colnames(data),
  tableLab = getLabelVar(tableVars, labelVars = labelVars),
  tableButton = TRUE,
  tablePars = list(),
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  labelVars = NULL,
  verbose = FALSE
)
```

Arguments

`data`	Data.frame with data.
`idVar`	String with variable containing subject ID.
`idLab`	String with label for `idVar`.
`keyVar`	String with unique key variable, identifying unique group for which the link between the table and the plot should be done.
`keyLab`	String with label for `keyVar`.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be:

	- compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot - included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`pathExpand`	Logical, should the variable in `pathExpand` be included in a collapsible row or as hyperlink in the table? Should be TRUE for if multiple paths are included for each `idVar`, FALSE otherwise (by default).
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`id`	String with general id for the plot: - 'id' is used as `group` for the `SharedData` - 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`labelVars`	Named character vector containing variable labels.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console.

Value

`datatable`

Author(s)

Laure Cougnaud

`timeProfileIntervalPlot`

Visualize time intervals across subjects/parameters.

Description

Visualize time intervals across subjects/parameters.

Usage

```
timeProfileIntervalPlot(
  data,
  paramVar,
  paramLab = getLabelVar(paramVar, labelVars = labelVars),
  paramVarSep = " - ",
  paramGroupVar = NULL,
  timeStartVar,
```

```
  timeStartLab = getLabelVar(timeStartVar, labelVars = labelVars),
  timeEndVar,
  timeEndLab = getLabelVar(timeEndVar, labelVars = labelVars),
  timeStartShapeVar = NULL,
  timeStartShapeLab = getLabelVar(timeStartShapeVar, labelVars = labelVars),
  timeEndShapeVar = NULL,
  timeEndShapeLab = getLabelVar(timeEndShapeVar, labelVars = labelVars),
  shapePalette = NULL,
  colorVar = NULL,
  colorLab = getLabelVar(colorVar, labelVars = labelVars),
  colorPalette = NULL,
  alpha = 1,
  yLab = NULL,
  xLab = paste(c(timeStartLab, timeEndLab), collapse = " and "),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  labelVars = NULL,
  width = 800,
  height = NULL,
  hoverVars,
  hoverLab,
  idVar = "USUBJID",
  idLab = getLabelVar(idVar, labelVars = labelVars),
  pathVar = NULL,
  pathLab = getLabelVar(pathVar, labelVars = labelVars),
  id = paste0("plotClinData", sample.int(n = 1000, size = 1)),
  selectVars = NULL,
  selectLab = getLabelVar(selectVars, labelVars = labelVars),
  table = FALSE,
  tableVars,
  tableLab,
  tableButton = TRUE,
  tablePars = list(),
  watermark = NULL,
  verbose = FALSE
)
```

Arguments

`data`	Data.frame with data.
`paramVar`	Character vector with variable of `data` to represent in the y-axis.
`paramLab`	(optional) String with label for `paramVar`.
`paramVarSep`	(optional) String with separator used to combined `paramVar` if multiple.
`paramGroupVar`	(optional) Character vector with variable(s) to group/order the `paramVar` elements in the y-axis.
`timeStartVar`	String with variable with the start of the time interval.

`timeStartLab`	(optional) String with label for `timeStartVar`.
`timeEndVar`	String with variable with the end of the time interval.
`timeEndLab`	(optional) String with label for `timeEndVar`.
`timeStartShapeVar`	(optional) String with variable used for the shape of the start of the time interval.
`timeStartShapeLab`	(optional) String with label for `timeStartShapeVar`.
`timeEndShapeVar`	(optional) String with variable used for the shape of the end of the time interval.
`timeEndShapeLab`	(optional) String with label for `timeEndShapeVar`.
`shapePalette`	(optional) Character vector with shape palette for `timeStartShapeVar` and `timeEndShapeVar`.
`colorVar`	(optional) String with color variable.
`colorLab`	String with label for `colorVar`.
`colorPalette`	(optional) Named character vector with color palette. If not specified, the viridis color palette is used. See `clinColors`.
`alpha`	(optional) Numeric with transparency, 1 by default.
`xLab`, `yLab`	(optional) String with labels for the x/y-axis.
`title`	String with title for the plot.
`subtitle`	String with subtitle. The subtitle is included at the top left of the plot, below the title.
`caption`	String with caption. The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.
`labelVars`	Named character vector containing variable labels.
`width`	Numeric, width of the plot in pixels, 800 by default.
`height`	Numeric, height of the plot in pixels, 500 by default.
`hoverVars`	Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.
`hoverLab`	Named character vector with labels for `hoverVars`.
`idVar`	String with variable containing subject ID.
`idLab`	String with label for `idVar`.
`pathVar`	String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as: `<a href="./path-to-report">label</a>` . If multiple, they should be separated by: ', '. The report(s) will be:

	• compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot • included in a collapsible row, and clickable with hyperlinks in the table
`pathLab`	String with label for `pathVar`, included in the collapsible row in the table.
`id`	String with general id for the plot: • 'id' is used as `group` for the `SharedData` • 'button:[id]' is used as button ID if `table` is TRUE If not specified, a random id, as 'plotClinData[X]' is used.
`selectVars`	(optional) Character vector with variable(s) from `data` for which a selection box should be included. This enables to select the data displayed in the plot (and associated table).
`selectLab`	(Named) character vector with label for `selectVars`.
`table`	Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. The plot and table are linked when included in a Rmarkdown document: when clicking on an plot element, only the corresponding records are retained in the associated table; when some records are selected in the table, they are highlighted in the associated table.
`tableVars`	Character vector with variables to be included in the table.
`tableLab`	Named character vector with labels for each `tableVars`.
`tableButton`	Logical, if TRUE (by default) the table is included within an HTML button.
`tablePars`	List with parameters passed to the `getClinDT` function.
`watermark`	(optional) String with path to a file containing a watermark.
`verbose`	Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other Clinical data visualization of individual profiles.: `scatterplotClinData()`

Examples

```
library(clinUtils)

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

dataAE <- dataADaMCDISCP01$ADAE

# basic plot
timeProfileIntervalPlot(
data = dataAE,
paramVar = "USUBJID",
# time-variables
timeStartVar = "ASTDY",
timeEndVar = "ASTDY",
# colored by severity
colorVar = "AESEV",
labelVars = labelVars
)

# add caption & subtitle
timeProfileIntervalPlot(
data = dataAE,
paramVar = "USUBJID",
timeStartVar = "ASTDY",
timeEndVar = "ASTDY",
colorVar = "AESEV",
labelVars = labelVars,
title = "Adverse events",
subtitle = "Time intervals",
caption = "Day is relative to the study baseline"
)

# add a selection box
if(interactive()){
  timeProfileIntervalPlot(
    data = dataAE,
    paramVar = "USUBJID",
    # time-variables
    timeStartVar = "ASTDY",
    timeEndVar = "ASTDY",
    # colored by severity
    colorVar = "AESEV",
    labelVars = labelVars,
    selectVars = "AEDECOD"
  )
}
```

`transformData` *Transform data.*

Description

Transform data from long to wide format. This function converts formats with the `stats::reshape` function.

Usage

```
transformData(data, transformations, verbose = FALSE, labelVars = NULL)
```

Arguments

`data` Data.frame with input data to transform.

`transformations`
Transformations (or list of those) as a list with:

- 'type': String with type of transformation. Currently, only: 'pivot_wider' is available
- extra parameters for the transformation, for:
    - 'pivot_wider':
        - 'varsID': Character vector with variable(s) of `data` defining unique records in the wide format. Corresponds to the `idvar` parameter of the `reshape` function.
        - 'varPivot': String with unique variable of `data` containing elements to pivot in different columns in the wide format (used for column names). Corresponds to the `timevar` parameter of the `reshape` function.
        - 'varsValue': Character vector with variable(s) of `data` used to fill the columns in the wide format. Corresponds to the `v.names` parameter of the `reshape` function.

`verbose` Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

`labelVars` Named character vector containing variable labels.

Value

A data.frame in wide format.

Author(s)

Laure Cougnaud

`treemapClinData` *Treemap interactive plot.*

Description

Note: the table and plot are not (yet) linked.

Usage

```
treemapClinData(...)
```

Arguments

`...`	Arguments passed on to `plotCountClinData`

`colorVar` (optional) String with coloring variable (NULL by default). By default, the treemap is colored based by section.

`colorRange` (optional) Numeric vector of length 2 with range for the color variable, in case it is a numeric variable.

`vars` Character vector with variables of `data` containing the groups. If multiple, they should be specified in hierarchical order (from parent to child node).

`varsLab` Named character vector with labels for `vars`.

`valueVar` String with numeric variable of `data` containing the value to display.

`valueLab` String with label for the `valueVar` variable.

`valueType` String with type of values in `valueVar` (`branchvalues` of the `plot_ly`) function), among others: 'total' (default, only if sum(child) <= to parent) or 'relative'.

`pathVar` String with variable of `data` containing hyperlinks with path to the subject-specific report, formatted as:

```
<a href="./path-to-report">label</a>
```

.

If multiple, they should be separated by: ', '.

The report(s) will be:

- compressed to a zip file and downloaded if the user clicks on the 'p' (a.k.a 'profile') key when hovering on a point of the plot
- included in a collapsible row, and clickable with hyperlinks in the table

`pathLab` String with label for `pathVar`, included in the collapsible row in the table.

`table` Logical, if TRUE (FALSE by default) returns also a `datatable` containing the plot data. (The plot and the table are not linked.)

`data` Data.frame with data.

`verbose` Logical, if TRUE (FALSE by default) progress messages are printed in the current console. For the visualizations, progress messages during download of subject-specific report are displayed in the browser console.

`width` Numeric, width of the plot in pixels, 800 by default.

`height` Numeric, height of the plot in pixels, 500 by default.

`hoverVars` Character vector with variable(s) to be displayed in the hover, by default any position (and axis) and aesthetic variables displayed in the plot.

`hoverLab` Named character vector with labels for `hoverVars`.

`labelVars` Named character vector containing variable labels.

`id` String with general id for the plot:

- 'id' is used as `group` for the `SharedData`
- 'button:[id]' is used as button ID if `table` is TRUE

If not specified, a random id, as 'plotClinData[X]' is used.

`title` String with title for the plot.

`titleExtra` String with extra title for the plot (appended after `title`).

`caption` String with caption.
The caption is included at the bottom right of the plot. Please note that this might overlap with vertical or rotated x-axis labels.

`subtitle` String with subtitle.
The subtitle is included at the top left of the plot, below the title.

`colorLab` String with label for `colorVar`.

`colorPalette` (optional) Named character vector with color palette. If not specified, the viridis color palette is used.
See `clinColors`.

`watermark` (optional) String with path to a file containing a watermark.

`tableButton` Logical, if TRUE (by default) the table is included within an HTML button.

`tableVars` Character vector with variables to be included in the table.

`tableLab` Named character vector with labels for each `tableVars`.

`tablePars` List with parameters passed to the `getClinDT` function.

Value

Either:

- if a `table` is requested: a `clinDataReview` object, a.k.a a list with the 'plot' (`plotly` object) and 'table' (`datatable` object)
- otherwise: a `plotly` object

Author(s)

Laure Cougnaud

See Also

Other visualizations of summary statistics for clinical data: `barplotClinData()`, `boxplotClinData()`, `errorbarClinData()`, `plotCountClinData()`, `sunburstClinData()`

Examples

```
library(clinUtils)

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

dataDM <- dataADaMCDISCP01$ADSL
dataAE <- dataADaMCDISCP01$ADAE

library(plyr)

## basic treemap:

# treemap takes as input table with counts
if (requireNamespace("inTextSummaryTable", quietly = TRUE)) {

# total counts: Safety Analysis Set (patients with start date for the first treatment)
dataTotal <- subset(dataDM, RFSTDTC != "")

# compute adverse event table
tableAE <- inTextSummaryTable::getSummaryStatisticsTable(

data = dataAE,
rowVar = c("AESOC", "AEDECOD"),
dataTotal = dataTotal,
rowOrder = "total",
labelVars = labelVars,
stats = inTextSummaryTable::getStats("count"),

# plotly treemap requires records (rows) for each group
rowVarTotalInclude = "AEDECOD",
outputType = "data.frame-base"

)

dataPlot <- tableAE

dataPlot$n <- as.numeric(dataPlot$n)

# create plot
treemapClinData(
data = dataPlot,
vars = c("AESOC", "AEDECOD"),
valueVar = "n",
    valueLab = "Number of patients with adverse events"
)

## treemap with coloring

# extract worst-case scenario
dataAE$AESEVN <- as.numeric(factor(dataAE$AESEV, levels = c("MILD", "MODERATE", "SEVERE")))
if(any(is.na(dataAE$AESEVN)))
```

```
stop("Severity should be filled for all subjects.")

dataAEWC <- ddply(dataAE, c("AESOC", "AEDECOD", "USUBJID"), function(x){
x[which.max(x$AESEVN), ]
})
dataTotalRow <- list(AEDECOD =
ddply(dataAEWC, c("AESOC", "USUBJID"), function(x){
x[which.max(x$AESEVN), ]
})
)


# compute adverse event table
tableAE <- inTextSummaryTable::getSummaryStatisticsTable(

data = dataAEWC,
rowVar = c("AESOC", "AEDECOD"),
var = "AESEVN",
dataTotal = dataTotal,
rowOrder = "total",
labelVars = labelVars,

# plotly treemap requires records (rows) for each group
rowVarTotalInclude = "AEDECOD",
dataTotalRow = dataTotalRow,
outputType = "data.frame-base"

)

dataPlot <- tableAE

dataPlot$statN <- as.numeric(dataPlot$statN)
dataPlot$statMean <- as.numeric(dataPlot$statMean)

# create plot
treemapClinData(
data = dataPlot,
vars = c("AESOC", "AEDECOD"),
valueVar = "statN", valueLab = "Number of patients with adverse events",
colorVar = "statMean", colorLab = "Mean severity"
)

}
```

`varToFm` *Get formula for a specific variable, to be used in aesthetic specification in* `plot_ly`*.*

Description

Get formula for a specific variable, to be used in aesthetic specification in `plot_ly`.

Usage

```
varToFm(var)
```

Arguments

`var`	Character vector with variable to combine. Otherwise with the '+' operator.

Value

`as.formula`

Author(s)

Laure Cougnaud

`zipClinDataReview` *Zip the clinical data report*

Description

Create a zip folder of clinical data reports with a redirect page. The clinical data report out of the `render_clinDataReviewReport` is copied into a new folder. A redirect html page is created to enable the user to navigate the report without needing to look into the new directory.

Usage

```
zipClinDataReview(
  reportDir = "report",
  newDir = "report_dependencies",
  redirectPage = "report.html",
  zipFolder = "report.zip"
)
```

Arguments

`reportDir`	String for the path to the directory where the clinical data reports are stored
`newDir`	String for the path where the files from `reportDir` should be copied to.
`redirectPage`	String with the path of the html file that redirects to the "1-introduction.html" page of the report.
`zipFolder`	String with the path to the zipped folder.

Value

The zip folder is created in the specified location.

Index