Package ‘safetyGraphics’

January 15, 2020

Title Create Interactive Graphics Related to Clinical Trial Safety

Version 1.1.0

Maintainer Jeremy Wildfire <jeremy_wildfire@rhoworld.com>

Description A framework for evaluation of clinical trial safety. Users can interactively explore their data using the 'Shiny' application or create standalone 'htmlwidget' charts. Interactive charts are built using 'd3.js' and 'webcharts.js' 'JavaScript' libraries.

URL https://github.com/SafetyGraphics/safetyGraphics

BugReports https://github.com/SafetyGraphics/safetyGraphics/issues

Depends R (>= 3.5)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests testthat, shinytest, knitr, ggplot2, plotly

Imports dplyr, htmlwidgets, purrr, stringr, jsonlite, shiny, magrittr, DT, shinyjs, rmarkdown, rlang, tibble, utils, haven, shinyWidgets, tidyr, shinybusy

VignetteBuilder knitr

NeedsCompilation no

Author Jeremy Wildfire [cre, aut],
Becca Krouse [aut],
Preston Burns [aut],
Xiao Ni [aut],
James Buchanan [aut],
Susan Duke [aut],
Interactive Safety Graphics Working Group [aut],
Rho Inc. [cph]

Repository CRAN

Date/Publication 2020-01-15 22:50:05 UTC

R topics documented:

addChart 2
addSetting 3
adlbc 4
chartRenderer 6
chartRenderer-shiny 7
chartsMetadata 7
detectStandard 8
generateSettings 9
getRequiredSettings 10
getSettingsMetadata 11
removeCharts 12
removeSettings 12
safetyGraphicsApp 13
settingsMetadata 13
standardsMetadata 14
trimSettings 15
validateSettings 16

Index **18**

`addChart` *Adds a new chart for use in the safetyGraphics shiny app*

Description

This function updates settings objects to add a new chart to the safetyGraphics shiny app

Usage

```
addChart(chart, label = "", description = "", repo_url = "",
  settings_url = "", main = "character", type = "static",
  maxWidth = 1000, requiredSettings = c(""),
  settingsLocation = getwd(), overwrite = TRUE)
```

Arguments

`chart`	Name of the chart - one word, all lower case
`label`	Nicely formatted name of the chart
`description`	Description of the chart
`repo_url`	Homepage for chart's code repository (if any)
`settings_url`	Homepage for chart's settings documentation
`main`	Name of the main function used to initialize the app. If the type is htmlwidgets, the js function must accept "location" and "settings" parameters (in that order) and have an .init() method, expecting a json data array. Otherwise, the r function should accept named data and settings parameters, and should be loaded in the user's namespace.

`type`	type of chart. Should be 'static', 'plotly' or 'module'
`maxWidth`	max width for the widget in pixels
`requiredSettings`	array of text_key values (matching those used in settingsMetadata) for the required settings for this chart
`settingsLocation`	path where the custom settings will be loaded/saved. If metadata is not found in that location, it will be read from the package (e.g. safetyGraphics::settingsMetadata), and then written to the specified location once the new chart has been added.
`overwrite`	overwrite any existing chart metadata? Note that having multiple charts with the same name is not supported and will cause unexpected results. default = true

Details

This function makes it easy for users to add a new chart to the safetyGraphics shiny app, by making updates to the underlying metadata used by the package. Specifically, the function adds a row to chartsMetadata.rda describing the chart and adds a column to settingsMetadata.rda specifying which settings are used with the chart. If new settings are needed for the chart, the user should call addSetting() for each new setting required.

`addSetting` *Adds a new setting for use in the safetyGraphics shiny app*

Description

This function updates settings objects to add a new setting parameter to the safetyGraphics shiny app

Usage

```
addSetting(text_key, label, description, setting_type,
  setting_required = FALSE, column_mapping = FALSE, column_type = NA,
  field_mapping = FALSE, field_column_key = "", setting_cat,
  default = "", charts = c(), settingsLocation = getwd(),
  overwrite = TRUE)
```

Arguments

`text_key`	Text key indicating the setting name. `'--'` delimiter indicates a nested setting
`label`	Label
`description`	Description
`setting_type`	Expected type for setting value. Should be "character", "vector", "numeric" or "logical"
`setting_required`	Flag indicating if the setting is required

`column_mapping`	Flag indicating if the setting corresponds to a column in the associated data
`column_type`	Expected type for the data column values. Should be "character","logical" or "numeric"
`field_mapping`	Flag indicating whether the setting corresponds to a field-level mapping in the data
`field_column_key`	Key for the column that provides options for the field-level mapping in the data
`setting_cat`	Setting category (data, measure, appearance)
`default`	Default value for non-data settings
`charts`	character vector of charts using this setting
`settingsLocation`	path where the custom settings will be loaded/saved. If metadata is not found in that location, it will be read from the package (e.g. safetyGraphics::settingsMetadata), and then written to the specified location once the new setting has been added.
`overwrite`	overwrite any existing setting metadata? Note that having settings with the same name is not supported and will cause unexpected results. default = true

Details

This function makes it easy for users to adds a new settings to the safetyGraphics shiny app by making updates to the underlying metadata used by the package. Specifically, the function adds a row to settingsMetadata.rda describing the setting.

`adlbc` *Safety measures sample data*

Description

A dataset containing anonymized lab data from a clinical trial in the CDISC ADaM format. The structure is 1 record per measure per visit per participant. See a full description of the ADaM data standard here.

Usage

```
adlbc
```

Format

A data frame with 10288 rows and 46 variables.

STUDYID Study Identifier

SUBJID Subject Identifier for the Study

USUBJID Unique Subject Identifier

TRTP Planned Treatment

TRTPN	Planned Treatment (N)
TRTA	Actual Treatment
TRTAN	Actual Treatment (N)
TRTSDT	Date of First Exposure to Treatment
TRTEDT	Date of Last Exposure to Treatment
AGE	Age
AGEGR1	Age Group
AGEGR1N	Age Group (N)
RACE	Race
RACEN	Race (N)
SEX	Sex
COMP24FL	Completers Flag
DSRAEFL	Discontinued due to AE?
SAFFL	Safety Population Flag
AVISIT	Analysis Visit
AVISITN	Analysis Visit (N)
ADY	Analysis Relative Day
ADT	Analysis Relative Date
VISIT	Visit
VISITNUM	Visit (N)
PARAM	Parameter
PARAMCD	Parameter Code
PARAMN	Parameter (N)
PARCAT1	Parameter Category
AVAL	Analysis Value
BASE	Baseline Value
CHG	Change from Baseline
A1LO	Analysis Normal Range Lower Limit
A1HI	Analysis Normal Range Upper Limit
R2A1LO	Ratio to Low limit of Analysis Range
R2A1HI	Ratio to High limit of Analysis Range
BR2A1LO	Base Ratio to Analysis Range 1 Lower Lim
BR2A1HI	Base Ratio to Analysis Range 1 Upper Lim
ANL01FL	Analysis Population Flag
ALBTRVAL	Amount Threshold Range
ANRIND	Analysis Reference Range Indicator
BNRIND	Baseline Reference Range Indicator
ABLFL	Baseline Record Flag
AENTMTFL	Analysis End Date Flag
LBSEQ	Lab Sequence Number
LBNRIND	Reference Range Indicator
LBSTRESN	Numeric Result/Finding in Std Units

Source

https://github.com/RhoInc/data-library

`chartRenderer` *Create an interactive graphics widget*

Description

This function creates an nice interactive widget. See the vignettes for more details regarding how to customize charts.

Usage

```
chartRenderer(data, debug_js = FALSE, settings = NULL, chart = NULL)
```

Arguments

`data`	A data frame containing the labs data. Data must be structured as one record per study participant per time point per lab measure.
`debug_js`	print settings in javascript before rendering chart. Default: `FALSE`.
`settings`	Optional list of settings arguments to be converted to JSON using `jsonlite::toJSON(settings,auto_u = TRUE,dataframe = "rows",null = "null")`. Default: `NULL`.
`chart`	name of the chart to render

Examples

```
## Not run:

## Create Histogram figure using a premade settings list
details_list <- list(
  list(value_col = "TRTP", label = "Treatment"),
  list(value_col = "SEX", label = "Sex"),
  list(value_col = "AGEGR1", label = "Age group")
)


filters_list <- list(
  list(value_col = "TRTA", label = "Treatment"),
  list(value_col = "SEX", label = "Sex"),
  list(value_col = "RACE", label = "RACE"),
  list(value_col = "AGEGR1", label = "Age group")
)

settingsl <- list(id_col = "USUBJID",
      value_col = "AVAL",
      measure_col = "PARAM",
      unit_col = "PARAMCD",
      normal_col_low = "A1LO",
      normal_col_high = "A1HI",
      details = details_list,
      filters = filters_list)

chartRenderer(data=adlbc, settings = settingsl, chart=safetyhistogram)


## End(Not run)
```

`chartRenderer-shiny` *Shiny bindings for chartRenderer*

Description

Output and render functions for using safetyhistogram within Shiny applications and interactive Rmd documents.

Usage

```
output_chartRenderer(outputId, width = "100%", height = "400px")

render_chartRenderer(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.
`expr`	An expression that generates a chart
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

`chartsMetadata` *Charts Metadata*

Description

Metadata about the charts available in the shiny app

Usage

```
chartsMetadata
```

Format

A data frame with 29 rows and 7 columns

chart Name of the chart - one word, all lower case

label Nicely formatted name of the chart

description Description of the chart

repo_url Homepage for chart's code repository (if any)

settings_url Homepage for chart's settings documentation

main Name of the main function used to initialize the app. The function must accept "location" and "settings" parameters (in that order) and have an .init() method, expecting a json data array.

type type of chart (e.g. 'htmlwidget')

maxWidth max width for the widget in pixels

Source

Created for this package

`detectStandard` *Detect the data standard used for a data set*

Description

This function attempts to detect the clinical data standard used in a given R data frame.

Usage

```
detectStandard(data, includeFields = TRUE, domain = "labs")
```

Arguments

`data`	A data frame in which to detect the data standard
`includeFields`	specifies whether to check the data set for field level data in addition to columns. Default: `TRUE`.
`domain`	The data domain for the data set provided. Default: `"labs"`.

Details

This function compares the columns in the provided `"data"` with the required columns for a given data standard/domain combination. The function is designed to work with the SDTM and ADaM CDISC(<https://www.cdisc.org/>) standards for clinical trial data by default. Additional standards can be added by modifying the `"standardMetadata"` data set included as part of this package. Currently, "labs" is the only domain supported.

Value

A list containing the matching "standard" from "standardMetadata" and a list of "details" describing each standard considered.

Examples

```
detectStandard(adlbc)[["standard"]] #adam
detectStandard(iris)[["standard"]] #none

## Not run:
 detectStandard(adlbc,domain="AE") #throws error. AE domain not supported in this release.

## End(Not run)
```

`generateSettings` *Generate a settings object based on a data standard*

Description

This function returns a settings object for the eDish chart based on the specified data standard.

Usage

```
generateSettings(standard = "None", charts = NULL,
  useDefaults = TRUE, partial = FALSE, partial_keys = NULL,
  custom_settings = NULL)
```

Arguments

`standard`	The data standard for which to create settings. Valid options are "sdtm", "adam" or "none". Default: `"None"`.
`charts`	The chart or charts for which settings should be generated. Default: `NULL` (uses all available charts).
`useDefaults`	Specifies whether default values from settingsMetadata should be included in the settings object. Default: `TRUE`.
`partial`	Boolean for whether or not the standard is a partial standard. Default: `FALSE`.
`partial_keys`	Optional character vector of the matched settings if partial is TRUE. Settings should be identified using the text_key format described in ?settingsMetadata. Setting is ignored when partial is FALSE. Default: `NULL`.
`custom_settings`	A tibble describing custom settings to be added to the settings object. Custom values overwrite default values when provided. Tibble should have text_key and customValue columns. Default: `NULL`.

Details

The function is designed to work with the SDTM and ADaM CDISC(<https://www.cdisc.org/>) standards for clinical trial data. Currently, eDish is the only chart supported.

Value

A list containing the appropriate settings for the selected chart

Examples

```
generateSettings(standard="SDTM")
generateSettings(standard="SdTm") #also ok
generateSettings(standard="ADaM")
pkeys<- c("id_col","measure_col","value_col")
generateSettings(standard="adam", partial=TRUE, partial_keys=pkeys)

generateSettings(standard="a different standard")
#returns shell settings list with no data mapping
```

`getRequiredSettings` *Get a list of required settings*

Description

Get a list of required settings for a given chart

Usage

```
getRequiredSettings(charts = NULL,
  metadata = safetyGraphics::settingsMetadata)
```

Arguments

`charts`	The chart for which required settings should be returned ("eDish" only for now) . Default: `NULL` (uses all available charts).
`metadata`	The metadata file to be used (primarily used for testing)

Value

List of lists specifying the position of matching named elements in the format `list("filters",2,"value_col")`, which would correspond to `settings[["filters"]][[2]][["value_col"]]`.

Examples

```
safetyGraphics:::getRequiredSettings(charts=c("edish","safetyHistogram"))
```

getSettingsMetadata *Get metadata about chart settings*

Description

Retrieve specified metadata about chart settings from the data/settingsMetadata.Rda file.

Usage

```
getSettingsMetadata(charts = NULL, text_keys = NULL, cols = NULL,
  filter_expr = NULL, add_standards = TRUE,
  metadata = safetyGraphics::settingsMetadata)
```

Arguments

`charts`	optional vector of chart names used to filter the metadata. Exact matches only (case-insensitive). All rows returned by default.
`text_keys`	optional vector of keys used to filter the metadata. Partial matches for any of the strings are returned (case-insensitive). All rows returned by default.
`cols`	optional vector of columns to return from the metadata. All columns returned by default.
`filter_expr`	optional filter expression used to subset the data.
`add_standards`	should data standard info stored in standardsMetadata be included
`metadata`	metadata data frame to be queried

Value

dataframe with the requested metadata or single metadata value

Examples

```
safetyGraphics:::getSettingsMetadata()
# Returns a full copy of settingsMetadata.Rda

safetyGraphics:::getSettingsMetadata(text_keys=c("id_col"))
# returns a dataframe with a single row with metadata for the id_col setting

safetyGraphics:::getSettingsMetadata(text_keys=c("id_col"), cols=c("label"))
# returns the character value for the specified row.
```

`removeCharts` *Remove a chart from the safetyGraphics shiny app*

Description

This function updates settings objects to remove chart from the safetyGraphics shiny app

Usage

```
removeCharts(charts, settingsLocation = getwd())
```

Arguments

`charts`	Name of the chart(s) to remove - one word, all lower case
`settingsLocation`	path where the custom settings will be loaded/saved. If metadata is not found in that location, it will be read from the package (e.g. safetyGraphics::settingsMetadata), and then written to the specified location once the chart has been removed

Details

This function makes it easy for remove a chart from the safetyGraphics shiny app by making updates to the underlying metadata used by the package.

`removeSettings` *Remove a setting from the safetyGraphics shiny app*

Description

This function updates settings objects to remove a setting parameter from the safetyGraphics shiny app

Usage

```
removeSettings(text_keys, settingsLocation = getwd())
```

Arguments

`text_keys`	Text keys indicating the setting names to be removed.
`settingsLocation`	path where the custom settings will be loaded/saved. If metadata is not found in that location, it will be read from the package (e.g. safetyGraphics::settingsMetadata), and then written to the specified location once the setting has been removed.

Details

This function makes it easy for remove a setting from the safetyGraphics shiny app by making updates to the underlying metadata used by the package.

`safetyGraphicsApp` *Run the interactive safety graphics builder*

Description

Run the interactive safety graphics builder

Usage

```
safetyGraphicsApp(charts = NULL, maxFileSize = NULL,
  settingsLocation = ".", customSettings = "customSettings.R",
  loadData = FALSE)
```

Arguments

`charts`	Character vector of charts to include
`maxFileSize`	maximum file size in MB allowed for file upload
`settingsLocation`	folder location of user-defined settings metadata. Files should be named settingsMetadata.rda, chartsMetadata.rda and standardsMetadata.rda and use the same structure established in the /data folder. Defaults to current working directory.
`customSettings`	Name of R script containing settings customizations to be run before the app is initialized. This is the recommended way to add additional charts (via addChart()), settings (addSetting()) and data standards (addStandard()). default = 'settingsLocation/customSettings.R'
`loadData`	Option to pre-load data into the app. Defaults to `FALSE`.

`settingsMetadata` *Settings Metadata*

Description

Metadata about the settings used to configure safetyGraphics charts. One record per unique setting

Usage

```
settingsMetadata
```

Format

A data frame with 29 rows and 17 columns

chart_hepexplorer Flag indicating if the settings apply to the Hepatic Explorer Chart

chart_paneledoutlierexplorer Flag indicating if the settings apply to the Paneled Safety Outlier Explorer Chart

chart_safetyhistogram Flag indicating if the settings apply to the Safety Histogram Chart

chart_safetyoutlierexplorer Flag indicating if the settings apply to the Safety Outlier Explorer Chart

chart_safetyresultsovertime Flag indicating if the settings apply to the Safety Results Over Time Chart

chart_safetyshiftplot Flag indicating if the settings apply to the Safety Shift Plot Chart

chart_safetydeltadelta Flag indicating if the settings apply to the Safety Delta-Delta Chart

text_key Text key indicating the setting name. `'--'` delimiter indicates a nested setting

label Label

description Description

setting_type Expected type for setting value. Should be "character", "vector", "numeric" or "logical"

setting_required Flag indicating if the setting is required

column_mapping Flag indicating if the setting corresponds to a column in the associated data

column_type Expected type for the data column values. Should be "character","logical" or "numeric"

field_mapping Flag indicating whether the setting corresponds to a field-level mapping in the data

field_column_key Key for the column that provides options for the field-level mapping in the data

setting_cat Setting category (data, measure, appearance)

default Default value for non-data settings

Source

Created for this package

`standardsMetadata` *Standards Metadata*

Description

Metadata about the data standards used to configure safetyGraphics charts. One record per unique setting. Columns contain default setting values for clinical data standards, like the CDISC "adam" and "sdtm" standards.

Usage

```
standardsMetadata
```

Format

A data frame with 25 rows and 3 columns

text_key Text key indicating the setting name. `'--'` delimiter indicates a nested setting

adam Settings values for the ADaM standard

sdtm Settings values for the SDTM standard

Source

Created for this package

`trimSettings` *Subset a settings object to those relevant for a list of charts*

Description

This function returns a settings object

Usage

```
trimSettings(settings, charts = NULL)
```

Arguments

`settings`	The settings list to subset
`charts`	The charts to subset by

Details

This function returns a settings object subsetted to the settings relevant to a vector of charts

Value

A list containing settings subsetted for the selected charts

Examples

```
testSettings <- generateSettings(standard="adam")
trimSettings(settings=testSettings, charts = c("safetyhistogram","edish"))
```

`validateSettings` *Compare a settings object with a specified data set*

Description

This function returns a list describing the validation status of a data set for a specified data standard

Usage

```
validateSettings(data, settings, charts = NULL)
```

Arguments

`data`	A data frame to check against the settings object
`settings`	The settings list to compare with the data frame.
`charts`	The charts being validated

Details

This function returns a list describing the validation status of a settings/data combo for a given chart type. This list can be used to populate status fields and control workflow in the Shiny app. It could also be used to manually QC a buggy chart. The tool checks that all setting properties containing "_col" match columns in the data set via `checkColumnSettings`, and all properties containing "_values" match fields in the data set via `checkFieldSettings`.

Value

A list describing the validation state for the data/settings combination. The returned list has the following properties:

- "valid" - boolean indicating whether the settings/data combo is valid for all charts
- "status" - string summarizing the validation results
- "charts" - a list of lists specifying whether each chart is valid. Each item in the list has "chart" and "valid" properties
- "checkList" - list of lists giving details about checks performed on individual setting specifications. Each embedded item has the following properties:
  - "key" - list specifying the position of the property being checked. For example, ‘list("group_cols",1,"value_col")‘ corresponds to ‘settings[["group_cols"]][[1]][["value_col"]]‘
  - "text_key" - list from ‘key‘ parsed to character with a "–" separator.
  - "value" - value of the setting
  - "type" - type of the check performed.
  - "description" - description of the check performed.
  - "valid" - boolean indicating whether the check was passed
  - "message" - string describing failed checks (where ‘valid=FALSE‘). returns an empty string when ‘valid==TRUE‘

Examples

```
testSettings <- generateSettings(standard="adam")
validateSettings(data=adlbc, settings=testSettings)
# .$valid is TRUE
testSettings$id_col <- "NotAColumn"
validateSettings(data=adlbc, settings=testSettings)
# .$valid is now FALSE
```

Index

∗Topic **datasets**
    adlbc, 4
    chartsMetadata, 7
    settingsMetadata, 13
    standardsMetadata, 14

addChart, 2
addSetting, 3
adlbc, 4

chartRenderer, 6
chartRenderer-shiny, 7
chartsMetadata, 7

detectStandard, 8

generateSettings, 9
getRequiredSettings, 10
getSettingsMetadata, 11

output_chartRenderer
        *(*chartRenderer-shiny*)*, 7

removeCharts, 12
removeSettings, 12
render_chartRenderer
        *(*chartRenderer-shiny*)*, 7

safetyGraphicsApp, 13
settingsMetadata, 13
standardsMetadata, 14

trimSettings, 15

validateSettings, 16