Package: teal.modules.general (via r-universe)

October 13, 2026

Type Package

Title General Modules for 'teal' Applications

Version 0.8.0.9003

Date 2026-10-08

Description Prebuilt 'shiny' modules containing tools for viewing data, visualizing data, understanding missing and outlier values within your data and performing simple data analysis. This extends 'teal' framework that supports reproducible research and analysis.

License Apache License 2.0

URL https://insightsengineering.github.io/teal.modules.general/, https://github.com/insightsengineering/teal.modules.general/

BugReports https://github.com/insightsengineering/teal.modules.general/issues

Depends ggplot2 (>= 3.5.0), R (>= 4.1), shiny (>= 1.8.1), teal (>= 1.2.0), teal.picks (>= 0.3.0), teal.transform (>= 0.7.0)

Imports broom (>= 0.7.10), bslib (>= 0.8.0), checkmate (>= 2.1.0), colourpicker (>= 1.3.0), dplyr (>= 1.1.0), DT (>= 0.13), forcats (>= 1.0.0), generics (>= 0.1.3), ggExtra (>= 0.10.1), ggpmisc (>= 0.5.6), ggpp (>= 0.5.8-1), ggrepel (>= 0.9.6), goftest (>= 1.2-3), grid, gridExtra (>= 2.3), htmlwidgets (>= 1.6.4), jsonlite (>= 1.8.9), lattice, lifecycle (>= 0.2.0), MASS (>= 7.3-60), patchwork (>= 1.2.0), rlang (>= 1.1.0), rmarkdown (>= 2.23), rtables (>= 0.6.11), scales (>= 1.3.0), shinyjs (>= 2.1.0), shinyTree (>= 0.2.8), shinyvalidate (>= 0.1.3), shinyWidgets (>= 0.5.1), sparkline (>= 2.0), stats, stringr (>= 1.4.1), teal.code (>= 0.7.0), teal.data (>= 0.8.0), teal.logger (>= 0.4.0), teal.reporter (>= 0.6.2), teal.widgets (>= 0.5.2), tern (>= 0.9.7), tibble (>= 2.0.0), tidyr (>= 0.8.3), tidyselect, tools, utils

Suggests gtsummary (>= 2.5.0), knitr (>= 1.42), logger (>= 0.4.0), nestcolor (>= 0.1.0), pkgload, roxy.shinylive, rvest, shinytest2, testthat (>= 3.1.9), withr (>= 2.0.0)

VignetteBuilder knitr, rmarkdown

Config/Needs/verdepcheck haleyjeppson/ggmosaic, tidyverse/ggplot2, rstudio/shiny, insightsengineering/teal, insightsengineering/teal.transform, mllg/checkmate, tidyverse/dplyr, rstudio/DT, tidyverse/forcats, r-lib/lifecycle, r-lib/scales, daattali/shinyjs, shinyTree/shinyTree, rstudio/shinyvalidate, dreamRs/shinyWidgets, tidyverse/stringr, insightsengineering/teal.code, insightsengineering/teal.data, insightsengineering/teal.logger, insightsengineering/teal.reporter, insightsengineering/teal.widgets, insightsengineering/tern, tidyverse/tibble, tidyverse/tidyr, daattali/colourpicker, daattali/ggExtra, aphalo/ggpmisc, aphalo/ggpp, slowkow/ggrepel, baddstats/goftest, ramnathv/htmlwidgets, jeroen/jsonlite, yihui/knitr, daroczig/logger, thomasp85/patchwork, insightsengineering/nestcolor, r-lib/pkgload, r-lib/rlang, rstudio/rmarkdown, insightsengineering/roxy.shinylive, insightsengineering/rtables, tidyverse/rvest, htmlwidgets/sparkline, rstudio/shinytest2, r-lib/testthat, r-lib/withr

Config/Needs/website insightsengineering/nesttemplate

Config/roxygen2/version 8.1.0

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE, packages = c(``roxy.shinylive''))

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libnode-dev zlib1g-dev

Repository https://insightsengineering.r-universe.dev

Date/Publication 2026-10-08 16:28:55 UTC

RemoteUrl https://github.com/insightsengineering/teal.modules.general

RemoteRef v0.9.0

RemoteSha c76d019fdee23c5f539644ca819bb0ca6b4932d3

Contents

add_facet_labels 3
tm_a_pca 4
tm_a_regression 8
tm_data_table 12
tm_file_viewer 15
tm_front_page 16
tm_g_association 18

tm_g_bivariate 22
tm_g_distribution 26
tm_g_response 30
tm_g_scatterplot 34
tm_g_scatterplotmatrix 39
tm_missing_data 44
tm_outliers 47
tm_rmarkdown 51
tm_t_crosstable 54
tm_tbl_summary 58
tm_variable_browser 63

Index **66**

`add_facet_labels` *Add labels for facets to a* `ggplot2` *object*

Description

Enhances a `ggplot2` plot by adding labels that describe the faceting variables along the x and y axes.

Usage

```
add_facet_labels(p, xfacet_label = NULL, yfacet_label = NULL)
```

Arguments

`p`	(`ggplot2`) object to which facet labels will be added.
`xfacet_label`	(`character`) Label for the facet along the x-axis. If `NULL`, no label is added. If a vector, labels are joined with " & ".
`yfacet_label`	(`character`) Label for the facet along the y-axis. Similar behavior to `xfacet_label`.

Value

Returns `grid` or `grob` object (to be drawn with `grid.draw`)

Examples

```
library(ggplot2)
library(grid)

p <- ggplot(mtcars) +
  aes(x = mpg, y = disp) +
  geom_point() +
  facet_grid(gear ~ cyl)

xfacet_label <- "cylinders"
yfacet_label <- "gear"
```

```
res <- add_facet_labels(p, xfacet_label, yfacet_label)
grid.newpage()
grid.draw(res)

grid.newpage()
grid.draw(add_facet_labels(p, xfacet_label = NULL, yfacet_label))
grid.newpage()
grid.draw(add_facet_labels(p, xfacet_label, yfacet_label = NULL))
grid.newpage()
grid.draw(add_facet_labels(p, xfacet_label = NULL, yfacet_label = NULL))
```

`tm_a_pca` `teal` *module: Principal component analysis*

Description

Module conducts principal component analysis (PCA) on a given dataset and offers different ways of visualizing the outcomes, including elbow plot, circle plot, biplot, and eigenvector plot. Additionally, it enables dynamic customization of plot aesthetics, such as opacity, size, and font size, through UI inputs.

Usage

```
tm_a_pca(
  label = "Principal Component Analysis",
  dat = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
     tidyselect::where(~is.numeric(.x) && all(!is.na(.x))), selected =
     tidyselect::everything(), multiple = TRUE)),
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  ggplot2_args = teal.widgets::ggplot2_args(),
  rotate_xaxis_labels = FALSE,
  font_size = c(12, 8, 20),
  alpha = c(1, 0, 1),
  size = c(2, 1, 8),
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`dat`	(`picks`) specifying columns used to compute PCA.

`plot_height` (`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.

`plot_width` (`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.

`ggtheme` (`character`) optional, `ggplot2` theme to be used by default. Defaults to `"gray"`.

`ggplot2_args` (`ggplot2_args`) optional, object created by `teal.widgets::ggplot2_args()` with settings for all the plots or named list of `ggplot2_args` objects for plot-specific settings. The argument is merged with options variable `teal.ggplot2_args` and default module setup.

List names should match the following: `c("default", "Elbow plot", "Circle plot", "Biplot", "Eigenvector plot")`.

For more details see the vignette: `vignette("custom-ggplot2-arguments", package = "teal.widgets")`.

`rotate_xaxis_labels` (`logical`) optional, whether to rotate plot X axis labels. Does not rotate by default (`FALSE`).

`font_size` (`numeric`) optional, specifies font size. It controls the font size for plot titles, axis labels, and legends.

- If vector of `length == 1` then the font sizes will have a fixed size.
- while vector of `value`, `min`, and `max` allows dynamic adjustment.

`alpha` (`integer(1)` or `integer(3)`) optional, specifies point opacity.

- When the length of `alpha` is one: the plot points will have a fixed opacity.
- When the length of `alpha` is three: the plot points opacity are dynamically adjusted based on vector of `value`, `min`, and `max`.

`size` (`integer(1)` or `integer(3)`) optional, specifies point size.

- When the length of `size` is one: the plot point sizes will have a fixed size.
- When the length of `size` is three: the plot points size are dynamically adjusted based on vector of `value`, `min`, and `max`.

`pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

`post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.

`transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

`decorators` **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `elbow_plot` (`ggplot`)
- `circle_plot` (`ggplot`)
- `biplot` (`ggplot`)
- `eigenvector_plot` (`ggplot`)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_a_pca(
   ..., # arguments for module
   decorators = list(
    elbow_plot = teal_transform_module(...), # applied to the `elbow_plot` output
    circle_plot = teal_transform_module(...), # applied to the `circle_plot` output
      biplot = teal_transform_module(...), # applied to the `biplot` output
    eigenvector_plot = teal_transform_module(...) # applied to the `eigenvector_plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  USArrests <- USArrests
})

app <- init(
  data = data,
  modules = modules(
    tm_a_pca(
      "PCA",
      dat = teal.picks::picks(
        datasets("USArrests"),
        teal.picks::variables(
          choices = c("Murder", "Assault", "UrbanPop", "Rape"),
          selected = c("Murder", "Assault"),
          multiple = TRUE
        )
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}


# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_a_pca(
      dat = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(
          choices = c("BMRKR1", "AGE", "EOSDY"),
          selected = c("BMRKR1", "AGE"),
          multiple = TRUE
        )
      )
    )
  )
)
```

```
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

tm_a_regression teal *module: Scatterplot and regression analysis*

Description

Module for visualizing regression analysis, including scatterplots and various regression diagnostics plots. It allows users to explore the relationship between a set of regressors and a response variable, visualize residuals, and identify outliers.

Usage

```
tm_a_regression(
  label = "Regression Analysis",
 regressor = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
    is.numeric, selected = tidyselect::last_col(), multiple = TRUE)),
  response,
  outlier,
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  alpha = c(1, 0, 1),
  size = c(2, 1, 8),
 ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  ggplot2_args = teal.widgets::ggplot2_args(),
  pre_output = NULL,
  post_output = NULL,
  default_plot_type = 1,
  default_outlier_label = "USUBJID",
  label_segment_threshold = c(0.5, 0, 10),
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`regressor`	(`picks`) Specification for regressor variables selection. Created using `teal.picks::picks()`, which allows selecting variables to use as regressors in the regression model. `teal.picks::variables(multiple = TRUE)` allowed.
`response`	(`picks`) Specification for response variable selection. Created using `teal.picks::picks()`, which allows selecting a single numeric variable to use as the response in the regression model. `teal.picks::variables(multiple = TRUE)` not allowed.

`outlier` (`picks`) Optional specification for outlier label variable selection. Created using `teal.picks::picks()`, which allows selecting a factor or character variable to label outlier points on the plots.

`plot_height` (`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.

`plot_width` (`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.

`alpha` (`integer(1)` or `integer(3)`) optional, specifies point opacity.

- When the length of `alpha` is one: the plot points will have a fixed opacity.
- When the length of `alpha` is three: the plot points opacity are dynamically adjusted based on vector of `value`, `min`, and `max`.

`size` (`integer(1)` or `integer(3)`) optional, specifies point size.

- When the length of `size` is one: the plot point sizes will have a fixed size.
- When the length of `size` is three: the plot points size are dynamically adjusted based on vector of `value`, `min`, and `max`.

`ggtheme` (`character`) optional, `ggplot2` theme to be used by default. Defaults to `"gray"`.

`ggplot2_args` (`ggplot2_args`) optional, object created by `teal.widgets::ggplot2_args()` with settings for all the plots or named list of `ggplot2_args` objects for plot-specific settings. The argument is merged with options variable `teal.ggplot2_args` and default module setup.

List names should match the following: `c("default", "Response vs Regressor", "Residuals vs Fitted", "Scale-Location", "Cook's distance", "Residuals vs Leverage", "Cook's dist vs Leverage")`.

For more details see the vignette: `vignette("custom-ggplot2-arguments", package = "teal.widgets")`.

`pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

`post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.

`default_plot_type`
(`numeric`) optional, defaults to "Response vs Regressor".

1. Response vs Regressor
2. Residuals vs Fitted
3. Normal Q-Q
4. Scale-Location
5. Cook's distance
6. Residuals vs Leverage
7. Cook's dist vs Leverage

`default_outlier_label`
(`character`) optional, default column selected to label outliers.

`label_segment_threshold`
: (`numeric(1)` or `numeric(3)`) Minimum distance between label and point on the plot that triggers the creation of a line segment between the two. This may happen when the label cannot be placed next to the point as it overlaps another label or point. The value is used as the `min.segment.length` parameter to the `ggrepel::geom_text_repel()` function.

  It can take the following forms:

  - `numeric(1)`: Fixed value used for the minimum distance and the slider is not presented in the UI.
  - `numeric(3)`: A slider is presented in the UI (under "Plot settings") to adjust the minimum distance dynamically.
    It takes the form of `c(value, min, max)` and it is passed to the `value_min_max` argument in `teal.widgets::optionalSliderInputValMinMax`.

`transformators`
: (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

`decorators`
: **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `plot` (`ggplot`)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_a_regression(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using regression plots" via `vignette("using-regression-plots", package = "teal.modules.general")`.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  CO2 <- CO2
})

app <- init(
  data = data,
  modules = modules(
    tm_a_regression(
      label = "Regression",
      response = teal.picks::picks(
        datasets("CO2"),
        teal.picks::variables(choices = "uptake", selected = "uptake")
      ),
      regressor = teal.picks::picks(
        datasets("CO2"),
     teal.picks::variables(choices = c("conc", "Treatment"), selected = "conc", multiple = TRUE)
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
```

```
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_a_regression(
      label = "Regression",
      response = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(choices = "BMRKR1", selected = "BMRKR1")
      ),
      regressor = teal.picks::picks(
        datasets("ADSL"),
     teal.picks::variables(choices = c("AGE", "SEX", "RACE"), selected = "AGE", multiple = TRUE)
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_data_table` `teal` *module: Data table viewer*

Description

Module provides a dynamic and interactive way to view `data.frames` in a `teal` application. It uses the `DT` package to display data tables in a paginated, searchable, and sortable format, which helps to enhance data exploration and analysis.

Usage

```
tm_data_table(
  label = "Data Table",
  variables_selected = list(),
  datasets_selected = deprecated(),
  datanames = if (missing(datasets_selected)) "all" else datasets_selected,
  dt_args = list(),
 dt_options = list(searching = FALSE, pageLength = 30, lengthMenu = c(5, 15, 30, 100),
    scrollX = TRUE),
  server_rendering = FALSE,
  pre_output = NULL,
  post_output = NULL,
```

```
  transformators = list()
)
```

Arguments

`label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

`variables_selected` (`named list`) Character vectors of the variables (i.e. columns) which should be initially shown for each dataset. Names of list elements should correspond to the names of the datasets available in the app. If no entry is specified for a dataset, the first six variables from that dataset will initially be shown.

`datasets_selected` (`character`) **[Deprecated]** A vector of datasets which should be shown and in what order. Use `datanames` instead.

`datanames` (`character`) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors:

- The keyword `"all"` includes all datasets available in the data passed to the teal application.
- `NULL` hides the sidebar panel completely.
- If `transformators` are specified, their `datanames` are automatically added to this `datanames` argument.

`dt_args` (`named list`) Additional arguments to be passed to `DT::datatable()` (must not include `data` or `options`).

`dt_options` (`named list`) The `options` argument to `DT::datatable`. By default `list(searching = FALSE, pageLength = 30, lengthMenu = c(5, 15, 30, 100), scrollX = TRUE)`

`server_rendering` (`logical`) should the data table be rendered server side (see `server` argument of `DT::renderDataTable()`)

`pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

`post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.

`transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

Details

The `DT` package has an option `DT.TOJSON_ARGS` to show `Inf` and `NA` in data tables. Configure the `DT.TOJSON_ARGS` option via `options(DT.TOJSON_ARGS = list(na = "string"))` before running the module. Note though that sorting of numeric columns with `NA`/`Inf` will be lexicographic not numerical.

Value

Object of class `teal_module` to be used in `teal` applications.

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  iris <- iris
})

app <- init(
  data = data,
  modules = modules(
    tm_data_table(
      variables_selected = list(
       iris = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")
      ),
      dt_args = list(caption = "IRIS Table Caption")
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_data_table(
    variables_selected = list(ADSL = c("STUDYID", "USUBJID", "SUBJID", "SITEID", "AGE", "SEX")),
      dt_args = list(caption = "ADSL Table Caption")
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

tm_file_viewer teal *module: File viewer*

Description

The file viewer module provides a tool to view static files. Supported formats include text formats, PDF, PNG APNG, JPEG SVG, WEBP, GIF and BMP.

Usage

```
tm_file_viewer(
  label = "File Viewer Module",
  input_path = list(`Current Working Directory` = ".")
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
input_path	(list) of the input paths, optional. Each element can be: Paths can be specified as absolute paths or relative to the running directory of the application. Default to the current working directory if not supplied.

Value

Object of class teal_module to be used in teal applications.

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
data <- teal_data()
data <- within(data, {
  data <- data.frame(1)
})

app <- init(
  data = data,
  modules = modules(
    tm_file_viewer(
      input_path = list(
        folder = system.file("sample_files", package = "teal.modules.general"),
     png = system.file("sample_files/sample_file.png", package = "teal.modules.general"),
     txt = system.file("sample_files/sample_file.txt", package = "teal.modules.general"),
        url = file.path(
```

```
            "https://www.fda.gov/files/drugs/published",
            "Portable-Document-Format-Specifications.pdf"
          )
        )
      )
    )
  )
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_front_page` `teal` *module: Front page*

Description

Creates a simple front page for `teal` applications, displaying introductory text, tables, additional `html` or `shiny` tags, and footnotes.

Usage

```
tm_front_page(
  label = "Front page",
  header_text = character(0),
  tables = list(),
  additional_tags = tagList(),
  footnotes = character(0),
  show_metadata = deprecated(),
  datanames = if (missing(show_metadata)) NULL else "all",
  transformators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`header_text`	(`character` vector) text to be shown at the top of the module, for each element, if named the name is shown first in bold as a header followed by the value. The first element's header is displayed larger than the others.
`tables`	(`named list` of `data.frames`) tables to be shown in the module.
`additional_tags`	(`shiny.tag.list` or `html`) additional `shiny` tags or `html` to be included after the table, for example to include an image, `tagList(tags$img(src = "image.png"))` or to include further `html`, `HTML("html text here")`.
`footnotes`	(`character` vector) of text to be shown at the bottom of the module, for each element, if named the name is shown first in bold, followed by the value.

show_metadata (logical) **[Deprecated]** indicating whether the metadata of the datasets be available on the module. Metadata shown automatically when datanames set.

datanames (character) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors:

- The keyword "all" includes all datasets available in the data passed to the teal application.
- NULL hides the sidebar panel completely.
- If transformators are specified, their datanames are automatically added to this datanames argument.

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

Value

Object of class teal_module to be used in teal applications.

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
  attr(ADSL, "metadata") <- list("Author" = "NEST team", "data_source" = "synthetic data")
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

table_1 <- data.frame(Info = c("A", "B"), Text = c("A", "B"))
table_2 <- data.frame(`Column 1` = c("C", "D"), `Column 2` = c(5.5, 6.6), `Column 3` = c("A", "B"))
table_3 <- data.frame(Info = c("E", "F"), Text = c("G", "H"))

table_input <- list(
  "Table 1" = table_1,
  "Table 2" = table_2,
  "Table 3" = table_3
)

app <- init(
  data = data,
  modules = modules(
    tm_front_page(
      header_text = c(
        "Important information" = "It can go here.",
        "Other information" = "Can go here."
      ),
      tables = table_input,
```

```
      additional_tags = HTML("Additional HTML or shiny tags go here <br>"),
      footnotes = c("X" = "is the first footnote", "Y is the second footnote")
    )
  )
) |>
  modify_header(tags$h1("Sample Application")) |>
  modify_footer(tags$p("Application footer"))

if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_g_association` `teal` *module: Stack plots of variables and show association with reference variable*

Description

Module provides functionality for visualizing the distribution of variables and their association with a reference variable. It supports configuring the appearance of the plots, including themes and whether to show associations.

Usage

```
tm_g_association(
  label = "Association",
  ref = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
    is.numeric | teal.picks::is_categorical(min.len = 2, max.len = 10), selected = 1L),
      teal.picks::values()),
  vars,
  show_association = TRUE,
  plot_height = c(600, 400, 5000),
  plot_width = NULL,
  distribution_theme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic",
      "void"),
  association_theme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic",
      "void"),
  pre_output = NULL,
  post_output = NULL,
  ggplot2_args = teal.widgets::ggplot2_args(),
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

ref (picks) Reference variable specification created using teal.picks::picks().

vars (picks) Variables to be associated with the reference variable, specified using teal.picks::picks().

show_association
(logical) optional, whether show association of vars with reference variable. Defaults to TRUE.

plot_height (numeric) optional, specifies the plot height as a three-element vector of value, min, and max intended for use with a slider UI element.

plot_width (numeric) optional, specifies the plot width as a three-element vector of value, min, and max for a slider encoding the plot width.

distribution_theme, association_theme
(character) optional, ggplot2 themes to be used by default. Default to "gray".

pre_output (shiny.tag) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

ggplot2_args (ggplot2_args) optional, object created by teal.widgets::ggplot2_args() with settings for all the plots or named list of ggplot2_args objects for plot-specific settings. The argument is merged with options variable teal.ggplot2_args and default module setup.

List names should match the following: c("default", "Bivariate1", "Bivariate2").

For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets").

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

decorators **[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- plot (grob created with ggplot2::ggplotGrob())

A Decorator is applied to the specific output using a named list of teal_transform_module objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_association(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using association plot" via `vignette("using-association-plot", package = "teal.modules.general")`.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  CO2 <- CO2
  factors <- names(Filter(isTRUE, vapply(CO2, is.factor, logical(1L))))
  CO2[factors] <- lapply(CO2[factors], as.character)
})

app <- init(
  data = data,
  modules = modules(
    tm_g_association(
      ref = teal.picks::picks(
        datasets("CO2"),
        teal.picks::variables(
```

```
            choices = c("Plant", "Type", "Treatment"),
            selected = "Plant"
          )
        ),
        vars = teal.picks::picks(
          datasets("CO2"),
          teal.picks::variables(
            choices = c("Plant", "Type", "Treatment"),
            selected = "Treatment",
            multiple = TRUE
          )
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }

  # CDISC data example
  data <- teal_data()
  data <- within(data, {
    require(nestcolor)
    ADSL <- teal.data::rADSL
  })
  join_keys(data) <- default_cdisc_join_keys[names(data)]

  app <- init(
    data = data,
    modules = modules(
      tm_g_association(
        ref = teal.picks::picks(
          datasets("ADSL"),
          teal.picks::variables(
         choices = c("SEX", "RACE", "COUNTRY", "ARM", "STRATA1", "STRATA2", "ITTFL", "BMRKR2"),
            selected = "RACE"
          )
        ),
        vars = teal.picks::picks(
          datasets("ADSL"),
          teal.picks::variables(
         choices = c("SEX", "RACE", "COUNTRY", "ARM", "STRATA1", "STRATA2", "ITTFL", "BMRKR2"),
            selected = "BMRKR2",
            multiple = TRUE
          )
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }
```

tm_g_bivariate `teal` *module: Univariate and bivariate visualizations*

Description

Module enables the creation of univariate and bivariate plots, facilitating the exploration of data distributions and relationships between two variables.

Usage

```
tm_g_bivariate(
  label = "Bivariate Plots",
  x = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
   is.numeric | teal.picks::is_categorical(min.len = 2, max.len = 10), selected = 1L),
    teal.picks::values()),
  y,
  row_facet,
  col_facet,
  facet,
  color = NULL,
  fill = NULL,
  size = NULL,
  use_density = FALSE,
  color_settings = FALSE,
  free_x_scales = FALSE,
  free_y_scales = FALSE,
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  rotate_xaxis_labels = FALSE,
  swap_axes = FALSE,
 ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  ggplot2_args = teal.widgets::ggplot2_args(),
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`x`	(`picks`) Variable specification for the x-axis. Created using `teal.picks::picks()`. Can be numeric, factor or character. No empty selections are allowed.
`y`	(`picks`) Variable specification for the y-axis. Created using `teal.picks::picks()`. Can be numeric, factor or character.

row_facet (picks) optional, specification of the data variable(s) to use for faceting rows. Created using teal.picks::picks().

col_facet (picks) optional, specification of the data variable(s) to use for faceting columns. Created using teal.picks::picks().

facet (logical) optional, specifies whether the facet encodings ui elements are toggled on and shown to the user by default. Defaults to TRUE if either row_facet or column_facet are supplied.

color (picks) optional, specification of the data variable(s) selected for the outline color inside the coloring settings. It will be applied when color_settings is set to TRUE. Created using teal.picks::picks().

fill (picks) optional, specification of the data variable(s) selected for the fill color inside the coloring settings. It will be applied when color_settings is set to TRUE. Created using teal.picks::picks().

size (picks) optional, specification of the data variable(s) selected for the size of geom_point plots inside the coloring settings. It will be applied when color_settings is set to TRUE. Created using teal.picks::picks().

use_density (logical) optional, indicates whether to plot density (TRUE) or frequency (FALSE). Defaults to frequency (FALSE).

color_settings (logical) Whether coloring, filling and size should be applied and UI tool offered to the user.

free_x_scales (logical) optional, whether X scaling shall be changeable. Does not allow scaling to be changed by default (FALSE).

free_y_scales (logical) optional, whether Y scaling shall be changeable. Does not allow scaling to be changed by default (FALSE).

plot_height (numeric) optional, specifies the plot height as a three-element vector of value, min, and max intended for use with a slider UI element.

plot_width (numeric) optional, specifies the plot width as a three-element vector of value, min, and max for a slider encoding the plot width.

rotate_xaxis_labels (logical) optional, whether to rotate plot X axis labels. Does not rotate by default (FALSE).

swap_axes (logical) optional, whether to swap X and Y axes. Defaults to FALSE.

ggtheme (character) optional, ggplot2 theme to be used by default. Defaults to "gray".

ggplot2_args (ggplot2_args) object created by teal.widgets::ggplot2_args() with settings for the module plot. The argument is merged with options variable teal.ggplot2_args and default module setup.

For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets")

pre_output (shiny.tag) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

decorators **[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Details

This is a general module to visualize 1 & 2 dimensional data.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- plot (ggplot)

A Decorator is applied to the specific output using a named list of teal_transform_module objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_bivariate(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette vignette("decorate-module-output", package = "teal.modules.general").

To learn more please refer to the vignette vignette("transform-module-output", package = "teal") or the teal::teal_transform_module() documentation.

Reporting

This module returns an object of class teal_module, that contains a server function. Since the server function returns a teal_report object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the teal framework.

For more information on reporting in teal, see the vignettes:

- vignette("reportable-shiny-application", package = "teal.reporter")
- vignette("adding-support-for-reporting-to-custom-modules", package = "teal")

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using bivariate plot" via `vignette("using-bivariate-plot", package = "teal.modules.general")`.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  CO2 <- data.frame(CO2)
})

app <- init(
  data = data,
  modules = tm_g_bivariate(
    label = "Bivariate Plots",
    x = teal.picks::picks(
      datasets("CO2"),
      teal.picks::variables(selected = "conc")
    ),
    y = teal.picks::picks(
      datasets("CO2"),
      teal.picks::variables(selected = "uptake")
    ),
    row_facet = teal.picks::picks(
      datasets("CO2"),
      teal.picks::variables(selected = "Type")
    ),
    col_facet = teal.picks::picks(
      datasets("CO2"),
      teal.picks::variables(selected = "Treatment")
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]
```

```
app <- init(
  data = data,
  modules = tm_g_bivariate(
    label = "Bivariate Plots",
    x = teal.picks::picks(
      datasets("ADSL"),
      teal.picks::variables(selected = "AGE")
    ),
    y = teal.picks::picks(
      datasets("ADSL"),
      teal.picks::variables(selected = "SEX")
    ),
    row_facet = teal.picks::picks(
      datasets("ADSL"),
      teal.picks::variables(selected = "ARM")
    ),
    col_facet = teal.picks::picks(
      datasets("ADSL"),
      teal.picks::variables(selected = "COUNTRY")
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_g_distribution` `teal` *module: Distribution analysis*

Description

Module is designed to explore the distribution of a single variable within a given dataset. It offers several tools, such as histograms, Q-Q plots, and various statistical tests to visually and statistically analyze the variable's distribution.

Usage

```
tm_g_distribution(
  label = "Distribution Module",
 dist_var = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(is.numeric),
    teal.picks::values()),
  strata_var = NULL,
  group_var = NULL,
  freq = FALSE,
 ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  ggplot2_args = teal.widgets::ggplot2_args(),
  bins = c(30L, 1L, 100L),
```

```
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
dist_var	(picks or list of multiple picks) Variable(s) for which the distribution will be analyzed.
strata_var	(picks or list of multiple picks) Categorical variable used to split the distribution analysis.
group_var	(picks or list of multiple picks) Variable used for faceting plot into multiple panels.
freq	(logical) optional, whether to display frequency (TRUE) or density (FALSE). Defaults to density (FALSE).
ggtheme	(character) optional, ggplot2 theme to be used by default. Defaults to "gray".
ggplot2_args	(ggplot2_args) optional, object created by teal.widgets::ggplot2_args() with settings for all the plots or named list of ggplot2_args objects for plot-specific settings. The argument is merged with options variable teal.ggplot2_args and default module setup. List names should match the following: c("default", "Histogram", "QQplot"). For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets").
bins	(integer(1) or integer(3)) optional, specifies the number of bins for the histogram. • When the length of bins is one: The histogram bins will have a fixed size based on the bins provided. • When the length of bins is three: The histogram bins are dynamically adjusted based on vector of value, min, and max. Defaults to c(30L, 1L, 100L).
plot_height	(numeric) optional, specifies the plot height as a three-element vector of value, min, and max intended for use with a slider UI element.
plot_width	(numeric) optional, specifies the plot width as a three-element vector of value, min, and max for a slider encoding the plot width.
pre_output	(shiny.tag) optional, with text placed before the output to put the output into context. For example a title.
post_output	(shiny.tag) optional, with text placed after the output to put the output into context. For example the shiny::helpText() elements are useful.

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

decorators **[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators::

- histogram_plot (ggplot)
- qq_plot (ggplot)
- summary_table (DT::datatable)
- test_table (DT::datatable)

A Decorator is applied to the specific output using a named list of teal_transform_module objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_distribution(
   ..., # arguments for module
   decorators = list(
   histogram_plot = teal_transform_module(...), # applied only to `histogram_plot` output
     qq_plot = teal_transform_module(...) # applied only to `qq_plot` output
   summary_table = teal_transform_module(...) # applied only to `summary_table` output
    test_table = teal_transform_module(...) # applied only to `test_table` output

   )
)
```

For additional details and examples of decorators, refer to the vignette vignette("decorate-module-output", package = "teal.modules.general").

To learn more please refer to the vignette vignette("transform-module-output", package = "teal") or the teal::teal_transform_module() documentation.

Reporting

This module returns an object of class teal_module, that contains a server function. Since the server function returns a teal_report object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the teal framework.

For more information on reporting in teal, see the vignettes:

- vignette("reportable-shiny-application", package = "teal.reporter")
- vignette("adding-support-for-reporting-to-custom-modules", package = "teal")

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  iris <- iris
})

app <- init(
  data = data,
  modules = list(
    tm_g_distribution(
      dist_var = teal.picks::picks(
        datasets("iris"),
        teal.picks::variables(is.numeric),
        teal.picks::values()
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_g_distribution(
      dist_var = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(c("BMRKR1", "AGE")),
        values(multiple = FALSE)
      ),
      strata_var = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(c("ARM", "COUNTRY", "SEX"), selected = NULL)
      ),
      group_var = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(c("ARM", "COUNTRY", "SEX"), selected = NULL)
      )
```

```
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_g_response` `teal` *module: Response plot*

Description

Generates a response plot for a given `response` and `x` variables. This module allows users customize and add annotations to the plot depending on the module's arguments. It supports showing the counts grouped by other variable facets (by row / column), swapping the coordinates, show count annotations and displaying the response plot as frequency or density.

Usage

```
tm_g_response(
  label = "Response Plot",
  response = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
    teal.picks::is_categorical(min.len = 2, max.len = 10)), teal.picks::values()),
  x,
  row_facet = NULL,
  col_facet = NULL,
  coord_flip = FALSE,
  count_labels = TRUE,
  rotate_xaxis_labels = FALSE,
  freq = FALSE,
  plot_height = c(600, 400, 5000),
  plot_width = NULL,
  ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  ggplot2_args = teal.widgets::ggplot2_args(),
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`response`	(`picks`) Which variable to use as the response. The `picks` must not allow multiple variable selection in this case.

x	(picks) Specifies which variable to use on the X-axis of the response plot. The picks must not allow multiple selection in this case.
row_facet	(picks) optional specification of the data variable(s) to use for faceting rows.
col_facet	(picks) optional specification of the data variable(s) to use for faceting columns.
coord_flip	(logical(1)) Indicates whether to flip coordinates between x and response. The default value is FALSE and it will show the x variable on the x-axis and the response variable on the y-axis.
count_labels	(logical(1)) Indicates whether to show count labels. Defaults to TRUE.
rotate_xaxis_labels	(logical) optional, whether to rotate plot X axis labels. Does not rotate by default (FALSE).
freq	(logical(1)) Indicates whether to display frequency (TRUE) or density (FALSE). Defaults to density (FALSE).
plot_height	(numeric) optional, specifies the plot height as a three-element vector of value, min, and max intended for use with a slider UI element.
plot_width	(numeric) optional, specifies the plot width as a three-element vector of value, min, and max for a slider encoding the plot width.
ggtheme	(character) optional, ggplot2 theme to be used by default. Defaults to "gray".
ggplot2_args	(ggplot2_args) object created by teal.widgets::ggplot2_args() with settings for the module plot. The argument is merged with options variable teal.ggplot2_args and default module setup. For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets")
pre_output	(shiny.tag) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.
post_output	(shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.
transformators	(list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").
decorators	**[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- plot (ggplot)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_response(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using response plot" via `vignette("using-response-plot", package = "teal.modules.general")`.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  mtcars <- mtcars
  for (.v in c("cyl", "vs", "am", "gear")) {
    mtcars[[.v]] <- as.factor(mtcars[[.v]])
  }
})

app <- init(
```

```
  data = data,
  modules = modules(
    tm_g_response(
      label = "Response Plots",
      response = teal.picks::picks(
        teal.picks::datasets("mtcars"),
        teal.picks::variables(
          choices = c("cyl", "gear"),
          selected = "cyl",
          multiple = FALSE,
          fixed = FALSE
        ),
        teal.picks::values()
      ),
      x = teal.picks::picks(
        datasets("mtcars"),
        teal.picks::variables(
          choices = c("vs", "am"),
          selected = "vs",
          multiple = FALSE,
          fixed = FALSE
        ),
        teal.picks::values()
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_g_response(
      label = "Response Plots",
      response = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(
          choices = c("BMRKR2", "COUNTRY"),
          selected = "BMRKR2"
        ),
        teal.picks::values()
      ),
      x = teal.picks::picks(
```

```
          datasets("ADSL"),
          teal.picks::variables(
            choices = c("SEX", "RACE"),
            selected = "RACE"
          ),
          teal.picks::values()
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }
```

`tm_g_scatterplot` `teal` *module: Scatterplot*

Description

Generates a customizable scatterplot using `ggplot2`. This module allows users to select variables for the x and y axes, color and size encodings, faceting options, and more. It supports log transformations, trend line additions, and dynamic adjustments of point opacity and size through UI controls.

Usage

```
tm_g_scatterplot(
  label = "Scatterplot",
  x = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(is.numeric),
     teal.picks::values()),
  y,
  color_by = NULL,
  size_by = NULL,
  row_facet = NULL,
  col_facet = NULL,
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  alpha = c(1, 0, 1),
  shape = shape_names,
  size = c(5, 1, 15),
  max_deg = 5L,
  rotate_xaxis_labels = FALSE,
  ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
  pre_output = NULL,
  post_output = NULL,
  table_dec = 4,
  ggplot2_args = teal.widgets::ggplot2_args(),
```

```
  transformators = list(),
  decorators = list()
)
```

Arguments

`label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

`x` (`picks` or `list` of multiple `picks`) Specifies variable names selected to plot along the x-axis by default.

`y` (`picks` or `list` of multiple `picks`) Specifies variable names selected to plot along the y-axis by default.

`color_by` (`picks` or `list` of multiple `picks`) optional, defines the color encoding. If `NULL` then no color encoding option will be displayed.

`size_by` (`picks` or `list` of multiple `picks`) optional, defines the point size encoding. If `NULL` then no size encoding option will be displayed.

`row_facet` (`picks` or `list` of multiple `picks`) optional, specifies the variable(s) for faceting rows.

`col_facet` (`picks` or `list` of multiple `picks`) optional, specifies the variable(s) for faceting columns.

`plot_height` (`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.

`plot_width` (`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.

`alpha` (`integer(1)` or `integer(3)`) optional, specifies point opacity.

- When the length of `alpha` is one: the plot points will have a fixed opacity.
- When the length of `alpha` is three: the plot points opacity are dynamically adjusted based on vector of `value`, `min`, and `max`.

`shape` (`character`) optional, character vector with the names of the shape, e.g. `c("triangle", "square", "circle")`. It defaults to `shape_names`. This is a complete list from `vignette("ggplot2-specs", package="ggplot2")`.

`size` (`integer(1)` or `integer(3)`) optional, specifies point size.

- When the length of `size` is one: the plot point sizes will have a fixed size.
- When the length of `size` is three: the plot points size are dynamically adjusted based on vector of `value`, `min`, and `max`.

`max_deg` (`integer`) optional, maximum degree for the polynomial trend line. Must not be less than 1.

`rotate_xaxis_labels` (`logical`) optional, whether to rotate plot X axis labels. Does not rotate by default (`FALSE`).

`ggtheme` (`character`) optional, `ggplot2` theme to be used by default. Defaults to `"gray"`.

`pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

table_dec (integer) optional, number of decimal places used to round numeric values in the table.

ggplot2_args (ggplot2_args) object created by teal.widgets::ggplot2_args() with settings for the module plot. The argument is merged with options variable teal.ggplot2_args and default module setup.
For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets")

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

decorators **[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- plot (ggplot)

A Decorator is applied to the specific output using a named list of teal_transform_module objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_scatterplot(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette vignette("decorate-module-output", package = "teal.modules.general").

To learn more please refer to the vignette vignette("transform-module-output", package = "teal") or the teal::teal_transform_module() documentation.

Reporting

This module returns an object of class teal_module, that contains a server function. Since the server function returns a teal_report object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the teal framework.

For more information on reporting in teal, see the vignettes:

- vignette("reportable-shiny-application", package = "teal.reporter")
- vignette("adding-support-for-reporting-to-custom-modules", package = "teal")

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using scatterplot" via vignette("using-scatterplot", package = "teal.modules.general").

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  CO2 <- CO2
})

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplot(
      label = "Scatterplot Choices",
      x = teal.picks::picks(
        datasets("CO2"),
        teal.picks::variables(
          choices = c("conc", "uptake"),
          selected = "conc"
        ),
        teal.picks::values()
      ),
      y = teal.picks::picks(
        datasets("CO2"),
        teal.picks::variables(
          choices = c("conc", "uptake"),
          selected = "uptake"
        ),
        teal.picks::values()
      ),
      color_by = teal.picks::picks(
        datasets("CO2"),
        teal.picks::variables(
          choices = c("Plant", "Type", "Treatment", "conc", "uptake"),
          selected = NULL
        ),
        teal.picks::values()
      ),
      size_by = teal.picks::picks(
```

```
          datasets("CO2"),
          teal.picks::variables(choices = c("conc", "uptake"), selected = "uptake"),
          teal.picks::values()
        ),
        row_facet = teal.picks::picks(
          datasets("CO2"),
          teal.picks::variables(
            choices = c("Plant", "Type", "Treatment"),
            selected = NULL
          ),
          teal.picks::values()
        ),
        col_facet = teal.picks::picks(
          datasets("CO2"),
         teal.picks::variables(choices = c("Plant", "Type", "Treatment"), selected = NULL),
          teal.picks::values()
        )
      )
    )
  )
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplot(
      label = "Scatterplot Choices",
      x = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(choices = c("AGE", "BMRKR1", "BMRKR2"), selected = "AGE"),
        teal.picks::values()
      ),
      y = teal.picks::picks(
        datasets("ADSL"),
       teal.picks::variables(choices = c("AGE", "BMRKR1", "BMRKR2"), selected = "BMRKR1"),
        teal.picks::values()
      ),
      color_by = teal.picks::picks(
        datasets("ADSL"),
      teal.picks::variables(c("AGE", "BMRKR1", "BMRKR2", "RACE", "REGION1"), selected = NULL),
        teal.picks::values()
      ),
      size_by = teal.picks::picks(
```

```
          datasets("ADSL"),
          teal.picks::variables(choices = c("AGE", "BMRKR1"), selected = "AGE"),
          teal.picks::values()
        ),
        row_facet = teal.picks::picks(
          datasets("ADSL"),
         teal.picks::variables(choices = c("BMRKR2", "RACE", "REGION1"), selected = NULL),
          teal.picks::values()
        ),
        col_facet = teal.picks::picks(
          datasets("ADSL"),
         teal.picks::variables(choices = c("BMRKR2", "RACE", "REGION1"), selected = NULL),
          teal.picks::values()
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }
```

---

tm_g_scatterplotmatrix

teal *module: Scatterplot matrix*

Description

Generates a scatterplot matrix from selected `variables` from datasets. Each plot within the matrix represents the relationship between two variables, providing the overview of correlations and distributions across selected data.

Usage

```
tm_g_scatterplotmatrix(
  label = "Scatterplot Matrix",
  variables = list(teal.picks::picks(teal.picks::datasets(),
    teal.picks::variables(selected = seq(1L, 5L), multiple = TRUE))),
  min_n_variables = 2L,
  max_n_variables = 5L,
  plot_height = c(600, 200, 2000),
  plot_width = NULL,
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

- `label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
- `variables` (`picks` or `list` of `picks`) Specifies plotting variables from an incoming dataset with filtering and selecting. In case of `picks` use `teal.picks::variables(..., ordered = TRUE)` if plot elements should be rendered according to selection order.
- `min_n_variables` (`integer(1)`) Minimum number of variables that must be selected.
- `max_n_variables` (`integer(1)`) Maximum number of variables that can be selected.
- `plot_height` (`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.
- `plot_width` (`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.
- `pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.
- `post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.
- `transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.
- `decorators` **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `plot` (`ggplot` - a `patchwork` assembled from individual `ggplot` panels)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_g_scatterplotmatrix(
   ..., # arguments for module
   decorators = list(
     plot = teal_transform_module(...) # applied to the `plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using scatterplot matrix" via `vignette("using-scatterplot-matrix", package = "teal.modules.general")`.

When *Add Correlation* is enabled, a simple **Omit NAs** checkbox controls NA handling (checked = `"pairwise.complete.obs"`, matching the historical default). Unchecking it reveals a dropdown with all five `stats::cor()` use options for advanced control.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  countries <- data.frame(
    id = c("DE", "FR", "IT", "ES", "PT", "GR", "NL", "BE", "LU", "AT"),
    government = factor(
      c(2, 2, 2, 1, 2, 2, 1, 1, 1, 2),
      labels = c("Monarchy", "Republic")
    ),
    language_family = factor(
      c(1, 3, 3, 3, 3, 2, 1, 1, 3, 1),
      labels = c("Germanic", "Hellenic", "Romance")
    ),
    population = c(83, 67, 60, 47, 10, 11, 17, 11, 0.6, 9),
    area = c(357, 551, 301, 505, 92, 132, 41, 30, 2.6, 83),
    gdp = c(3.4, 2.7, 2.1, 1.4, 0.3, 0.2, 0.7, 0.5, 0.1, 0.4),
    debt = c(2.1, 2.3, 2.4, 2.6, 2.3, 2.4, 2.3, 2.4, 2.3, 2.4)
  )
  sales <- data.frame(
    id = 1:50,
```

```
    country_id = sample(
      c("DE", "FR", "IT", "ES", "PT", "GR", "NL", "BE", "LU", "AT"),
      size = 50,
      replace = TRUE
    ),
    year = sort(sample(2010:2020, 50, replace = TRUE)),
    venue = sample(c("small", "medium", "large", "online"), 50, replace = TRUE),
    cancelled = sample(c(TRUE, FALSE), 50, replace = TRUE),
    quantity = rnorm(50, 100, 20),
    costs = rnorm(50, 80, 20),
    profit = rnorm(50, 20, 10)
  )
})
join_keys(data) <- join_keys(
  join_key("countries", "countries", "id"),
  join_key("sales", "sales", "id"),
  join_key("countries", "sales", c("id" = "country_id"))
)

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplotmatrix(
      label = "Scatterplot matrix",
      variables = list(
        teal.picks::picks(
          datasets("countries"),
          teal.picks::variables(
            choices = tidyselect::everything(),
            selected = c("area", "gdp", "debt"),
            multiple = TRUE,
            ordered = TRUE
          ),
          teal.picks::values()
        ),
        teal.picks::picks(
          datasets("sales"),
          teal.picks::variables(
            choices = c("quantity", "costs", "profit"),
            selected = c("quantity", "costs"),
            multiple = TRUE,
            ordered = TRUE
          )
        )
      ),
      transformators = list(
        teal_transform_filter(
          teal.picks::picks(
            datasets("sales"),
            teal.picks::variables("country_id"),
            teal.picks::values()
          )
        )
```

```
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }

  # CDISC data example
  data <- teal_data()
  data <- within(data, {
    ADSL <- teal.data::rADSL
    ADRS <- teal.data::rADRS
  })
  join_keys(data) <- default_cdisc_join_keys[names(data)]

  app <- init(
    data = data,
    modules = modules(
      tm_g_scatterplotmatrix(
        label = "Scatterplot matrix",
        variables = list(
          teal.picks::picks(
            datasets("ADSL"),
            teal.picks::variables(
              choices = tidyselect::everything(),
              selected = c("AGE", "RACE", "SEX"),
              multiple = TRUE,
              ordered = TRUE,
              fixed = FALSE
            ),
            teal.picks::values()
          ),
          teal.picks::picks(
            datasets("ADRS"),
            teal.picks::variables(
              choices = tidyselect::everything(),
              selected = c("AVAL", "ADY"),
              multiple = TRUE,
              ordered = TRUE,
              fixed = FALSE
            )
          )
        ),
        transformators = list(
          teal_transform_filter(
            teal.picks::picks(
              teal.picks::datasets("ADRS"),
              teal.picks::variables("PARAMCD"),
              teal.picks::values(selected = "BESRSPI")
            )
          )
        )
```

```
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

tm_missing_data teal *module: Missing data analysis*

Description

This module analyzes missing data in `data.frames` to help users explore missing observations and gain insights into the completeness of their data. It is useful for clinical data analysis within the context of `CDISC` standards and adaptable for general data analysis purposes.

Usage

```
tm_missing_data(
  label = "Missing data",
  plot_height = c(600, 400, 5000),
  plot_width = NULL,
  datanames = "all",
  parent_dataname = "ADSL",
 ggtheme = c("classic", "gray", "bw", "linedraw", "light", "dark", "minimal", "void"),
  ggplot2_args = list(`Combinations Hist` = teal.widgets::ggplot2_args(labs =
   list(caption = NULL)), `Combinations Main` = teal.widgets::ggplot2_args(labs =
    list(title = NULL))),
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`plot_height`	(`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.
`plot_width`	(`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.
`datanames`	(`character`) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors: • The keyword `"all"` includes all datasets available in the data passed to the teal application.

- NULL hides the sidebar panel completely.
- If transformators are specified, their datanames are automatically added to this datanames argument.

parent_dataname
: (character(1)) Specifies the parent dataset name. Default is ADSL for CDISC data. If provided and exists, enables additional analysis "by subject". For non-CDISC data, this parameter can be ignored.

ggtheme
: (character) optional, specifies the default ggplot2 theme for plots. Defaults to classic.

ggplot2_args
: (ggplot2_args) optional, object created by teal.widgets::ggplot2_args() with settings for all the plots or named list of ggplot2_args objects for plot-specific settings. The argument is merged with options variable teal.ggplot2_args and default module setup.

  List names should match the following: c("default", "Summary Obs", "Summary Patients", "Combinations Main", "Combinations Hist", "By Subject").

  For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets").

pre_output
: (shiny.tag) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output
: (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

transformators
: (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

decorators
: **[Experimental]** (named list of lists of teal_transform_module) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class teal_module to be used in teal applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- summary_plot (ggplot)
- combination_plot (grob created with ggplot2::ggplotGrob())
- by_variable_plot (ggplot)
- by_subject_plot (ggplot)

A Decorator is applied to the specific output using a named list of teal_transform_module objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_missing_data(
   ..., # arguments for module
   decorators = list(
    summary_plot = teal_transform_module(...), # applied only to `summary_plot` output
    combination_plot = teal_transform_module(...), # applied only to `combination_plot` output
    by_variable_plot = teal_transform_module(...) # applied only to `by_variable_plot` output
    by_subject_plot = teal_transform_module(...) # applied only to `by_subject_plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general example data
data <- teal_data()
data <- within(data, {
  require(nestcolor)

  add_nas <- function(x) {
    x[sample(seq_along(x), floor(length(x) * runif(1, .05, .17)))] <- NA
    x
  }

  iris <- iris
  mtcars <- mtcars

  iris[] <- lapply(iris, add_nas)
  mtcars[] <- lapply(mtcars, add_nas)
  mtcars[["cyl"]] <- as.factor(mtcars[["cyl"]])
  mtcars[["gear"]] <- as.factor(mtcars[["gear"]])
})
```

```
app <- init(
  data = data,
  modules = modules(
    tm_missing_data(parent_dataname = "mtcars")
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC example data
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- teal.data::rADSL
  ADRS <- rADRS
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_missing_data()
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_outliers` `teal` *module: Outliers analysis*

Description

Module to analyze and identify outliers using different methods such as IQR, Z-score, and Percentiles, and offers visualizations including box plots, density plots, and cumulative distribution plots to help interpret the outliers.

Usage

```
tm_outliers(
  label = "Outliers Module",
  outlier_var = teal.picks::picks(teal.picks::datasets(),
    teal.picks::variables(is.numeric, 1L, multiple = FALSE)),
  categorical_var = teal.picks::picks(teal.picks::datasets(),
   teal.picks::variables(choices = teal.picks::is_categorical(min.len = 1, max.len =
    10), selected = 1L, multiple = TRUE)),
```

```
  ggtheme = c("gray", "bw", "linedraw", "light", "dark", "minimal", "classic", "void"),
   ggplot2_args = teal.widgets::ggplot2_args(),
   plot_height = c(600, 200, 2000),
   plot_width = NULL,
   pre_output = NULL,
   post_output = NULL,
   transformators = list(),
   decorators = list()
)
```

Arguments

`label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

`outlier_var` (`picks` or `list` of multiple `picks`) Specifies variable(s) to be analyzed for outliers.

`categorical_var` (`picks` or `list` of multiple `picks`) optional, specifies the categorical variable(s) to split the selected outlier variables on.

`ggtheme` (`character`) optional, `ggplot2` theme to be used by default. Defaults to `"gray"`.

`ggplot2_args` (`ggplot2_args`) optional, object created by `teal.widgets::ggplot2_args()` with settings for all the plots or named list of `ggplot2_args` objects for plot-specific settings. The argument is merged with options variable `teal.ggplot2_args` and default module setup.

List names should match the following: `c("default", "Boxplot", "Density Plot", "Cumulative Distribution Plot")`.

For more details see the vignette: `vignette("custom-ggplot2-arguments", package = "teal.widgets")`.

`plot_height` (`numeric`) optional, specifies the plot height as a three-element vector of `value`, `min`, and `max` intended for use with a slider UI element.

`plot_width` (`numeric`) optional, specifies the plot width as a three-element vector of `value`, `min`, and `max` for a slider encoding the plot width.

`pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

`post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.

`transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

`decorators` **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `box_plot` (`ggplot`)
- `density_plot` (`ggplot`)
- `cumulative_plot` (`ggplot`)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_outliers(
   ..., # arguments for module
   decorators = list(
     box_plot = teal_transform_module(...), # applied only to `box_plot` output
    density_plot = teal_transform_module(...), # applied only to `density_plot` output
    cumulative_plot = teal_transform_module(...) # applied only to `cumulative_plot` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, CO2 <- CO2)

app <- init(
  data = data,
  modules = modules(
    tm_outliers(
      outlier_var = teal.picks::picks(
        datasets("CO2", "CO2"),
        teal.picks::variables(c("conc", "uptake"), "uptake"),
        teal.picks::values()
      ),
      categorical_var = teal.picks::picks(
        datasets("CO2", "CO2"),
        teal.picks::variables(c("Plant", "Type", "Treatment"), "Plant"),
        teal.picks::values()
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}


# CDISC data example
data <- teal_data()
data <- within(data, {
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_outliers(
      outlier_var = teal.picks::picks(
        datasets("ADSL", "ADSL"),
        teal.picks::variables(c("AGE", "BMRKR1"), "AGE")
      ),
      categorical_var = teal.picks::picks(
        datasets("ADSL", "ADSL"),
        teal.picks::variables(teal.picks::is_categorical(min.len = 1, max.len = 10))
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

tm_rmarkdown `teal` *module: R Markdown render*

Description

[Experimental]

Module to render R Markdown files using the data provided in the `teal_data` object.

The R Markdown file should be designed to accept variables available in the data names of the module.

Usage

```
tm_rmarkdown(
  label = "RMarkdown Module",
  rmd_content,
  datanames = "all",
  allow_download = TRUE,
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  extra_transform = list()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`rmd_content`	(`character`) Content of the R Markdown file to be rendered. This can be the value of `readLines("path/to/file.Rmd")`.
`datanames`	(`character`) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors: • The keyword `"all"` includes all datasets available in the data passed to the teal application. • `NULL` hides the sidebar panel completely. • If `transformators` are specified, their `datanames` are automatically added to this `datanames` argument.
`allow_download`	(`logical`) whether to allow downloading of the R Markdown file. Defaults to `TRUE`.
`pre_output`	(`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

transformators (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

extra_transform (list) of teal::teal_transform_module() that will be added in the module's UI. This can be used to create interactive inputs that modify the parameters in R Markdown rendering.

Details

For example, if the teal_data object contains datasets named mtcars and iris, the R Markdown file can use these as variables as they will be available in the R Markdown environment.

The libraries used in the R Markdown file must be available in the deployed shiny app environment.

When developing the R Markdown file, the working data can be simulated on a code chunk, which in turn can look for the presence of .raw_data object to determine if it is being run inside the teal module or not.

Example R markdown file:

```
---
title: "R Markdown Report"
output: html_document
---

```{r eval=!exists(".raw_data")}
mtcars <- datasets::mtcars
iris <- datasets::iris
```

```{r}
summary(mtcars) |> print()
summary(iris) |> print()
```
```

Value

Object of class teal_module to be used in teal applications.

Reporting

This module returns an object of class teal_module, that contains a server function. Since the server function returns a teal_report object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the teal framework.

For more information on reporting in teal, see the vignettes:

- vignette("reportable-shiny-application", package = "teal.reporter")
- vignette("adding-support-for-reporting-to-custom-modules", package = "teal")

Examples in Shinylive

example-1 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  CO2 <- CO2
})

app <- init(
  data = data,
  modules = modules(
    tm_rmarkdown(
      label = "RMarkdown Module",
      rmd_content = c(
        "---",
        "title: \"R Markdown Report\"",
        "output: html_document",
        "---",
        "",
        "```{r}",
        "summary(CO2) |> print()",
        "```"
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

nrow_transform <- teal_transform_module(
  label = "N Rows selector",
  ui = function(id) {
    ns <- NS(id)
    tags$div(
      numericInput(ns("n_rows"), "Show n rows", value = 40, min = 0, max = 200, step = 5)
    )
  },
  server = function(id, data) {
    moduleServer(id, function(input, output, session) {
      reactive({
        req(data())
        within(data(),
          {
            n_rows <- n_rows_value
          },
          n_rows_value = input$n_rows
        )
      })
```

```
    })
  }
)

app <- init(
  data = data,
  modules = modules(
    tm_rmarkdown(
      label = "RMarkdown Module",
      rmd_content = readLines(
        system.file(
          file.path("sample_files", "co2_example.Rmd"),
          package = "teal.modules.general"
        )
      ),
      allow_download = FALSE,
      extra_transform = list(nrow_transform)
    )
  )
)

if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

`tm_t_crosstable` `teal` *module: Cross-table*

Description

Generates a simple cross-table of two variables from a dataset with custom options for showing percentages and sub-totals.

Usage

```
tm_t_crosstable(
  label = "Cross Table",
  x = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(choices =
   teal.picks::is_categorical(min.len = 2, max.len = 10), selected = 1L, multiple =
    TRUE, ordered = TRUE)),
  y,
  show_percentage = TRUE,
  show_total = TRUE,
  remove_zero_columns = FALSE,
  pre_output = NULL,
  post_output = NULL,
  basic_table_args = teal.widgets::basic_table_args(),
  transformators = list(),
  decorators = list()
)
```

Arguments

- `label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
- `x` (`picks` or `list` of `picks`) Object with all available choices with pre-selected option for variable X - row values. In case of `picks` use `teal.picks::variables(..., ordered = TRUE)` if table elements should be rendered according to selection order.
- `y` (`picks` or `list` of multiple `picks`) Object with all available choices with pre-selected option for variable Y - column values.

  `picks` must not allow multiple selection in this case.
- `show_percentage` (`logical(1)`) Indicates whether to show percentages (relevant only when `x` is a `factor`). Defaults to `TRUE`.
- `show_total` (`logical(1)`) Indicates whether to show total column. Defaults to `TRUE`.
- `remove_zero_columns` (`logical(1)`) Indicates whether to remove columns that contain only zeros from the output table. Defaults to `FALSE`.
- `pre_output` (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.
- `post_output` (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.
- `basic_table_args` (`basic_table_args`) object created by `teal.widgets::basic_table_args()` with settings for the module table. The argument is merged with options variable `teal.basic_table_args` and default module setup.

  For more details see the vignette: `vignette("custom-basic-table-arguments", package = "teal.widgets")`
- `transformators` (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.
- `decorators` **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Table Settings

The module provides several table settings that can be adjusted:

- `Show column percentage`: Shows column percentages when enabled
- `Show total column`: Shows a total column when enabled
- `Remove zero-only columns`: Removes columns that contain only zeros from the output table

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `table` (`ElementaryTable` - output of `rtables::build_table`)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_t_crosstable(
   ..., # arguments for module
   decorators = list(
     table = teal_transform_module(...) # applied to the `table` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

Reporting

This module returns an object of class `teal_module`, that contains a `server` function. Since the server function returns a `teal_report` object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the `teal` framework.

For more information on reporting in `teal`, see the vignettes:

- `vignette("reportable-shiny-application", package = "teal.reporter")`
- `vignette("adding-support-for-reporting-to-custom-modules", package = "teal")`

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Note

For more examples, please see the vignette "Using cross table" via `vignette("using-cross-table", package = "teal.modules.general")`.

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  mtcars <- mtcars
  for (v in c("cyl", "vs", "am", "gear")) {
    mtcars[[v]] <- as.factor(mtcars[[v]])
```

```
  }
  mtcars[["primary_key"]] <- seq_len(nrow(mtcars))
})
join_keys(data) <- join_keys(join_key("mtcars", "mtcars", "primary_key"))

app <- init(
  data = data,
  modules = modules(
    tm_t_crosstable(
      label = "Cross Table",
      x = teal.picks::picks(
        datasets("mtcars"),
        teal.picks::variables(
          choices = variable_choices(data[["mtcars"]], c("cyl", "vs", "am", "gear")),
          selected = c("cyl", "gear"),
          multiple = TRUE,
          ordered = TRUE,
          fixed = FALSE
        ),
        teal.picks::values()
      ),
      y = teal.picks::picks(
        datasets("mtcars"),
        teal.picks::variables(
          choices = variable_choices(data[["mtcars"]], c("cyl", "vs", "am", "gear")),
          selected = "vs",
          multiple = FALSE,
          fixed = FALSE
        ),
        teal.picks::values()
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  ADSL <- teal.data::rADSL
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_t_crosstable(
      label = "Cross Table",
      x = teal.picks::picks(
        datasets("ADSL"),
        teal.picks::variables(
```

```
            choices = variable_choices(data[["ADSL"]], subset = function(data) {
              idx <- !vapply(data, inherits, logical(1), c("Date", "POSIXct", "POSIXlt"))
              return(names(data)[idx])
            }),
            selected = "COUNTRY",
            multiple = TRUE,
            ordered = TRUE,
            fixed = FALSE
          ),
          teal.picks::values()
        ),
        y = teal.picks::picks(
          datasets("ADSL"),
          teal.picks::variables(
            choices = variable_choices(data[["ADSL"]], subset = function(data) {
              idx <- vapply(data, is.factor, logical(1))
              return(names(data)[idx])
            }),
            selected = "SEX",
            multiple = FALSE,
            fixed = FALSE
          ),
          teal.picks::values()
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }
```

`tm_tbl_summary` `teal` *module: GT Summary table*

Description

Summary table from a given dataset, using `gtsummary`.

Usage

```
tm_tbl_summary(
  label = "Summary table",
  by = NULL,
 include = teal.picks::picks(teal.picks::datasets(), teal.picks::variables(selected =
    dplyr::everything(), multiple = TRUE)),
  dataname = NULL,
  ...,
  col_label = NULL,
```

```
  pre_output = NULL,
  post_output = NULL,
  transformators = list(),
  decorators = list()
)
```

Arguments

`label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

`by` (`picks` or dplyr::dplyr_tidy_select) When using picks object it should have `teal.picks::variables()` to identify a single column from the data.

`include` (`picks` or dplyr::dplyr_tidy_select) When using picks object it should have `teal.picks::variables()` to identify one or more columns from the data.

`dataname` (`string` or `NULL`) Name of the dataset to be used in the module if and only if `picks` is not used for other arguments.

`...` Arguments passed on to `gtsummary::tbl_summary`

`statistic` (`formula-list-selector`)
: Specifies summary statistics to display for each variable. The default is `list(all_continuous() ~ "{median} ({p25}, {p75})", all_categorical() ~ "{n} ({p}%)")`. See below for details.

`digits` (`formula-list-selector`)
: Specifies how summary statistics are rounded. Values may be either integer(s) or function(s). If not specified, default formatting is assigned via `assign_summary_digits()`. See below for details.

`type` (`formula-list-selector`)
: Specifies the summary type. Accepted values are `c("continuous", "continuous2", "categorical", "dichotomous")`. If not specified, default type is assigned via `assign_summary_type()`. See below for details.

`value` (`formula-list-selector`)
: Specifies the level of a variable to display on a single row. The gtsummary type selectors, e.g. `all_dichotomous()`, cannot be used with this argument. Default is `NULL`. See below for details.

`missing,missing_text,missing_stat` Arguments dictating how and if missing values are presented:

- `missing`: specifies whether to include a row of missing/`NA` counts. Provide a (`formula-list-selector`) where each variable is assigned one of `c("ifany", "no", "always")` (e.g. `missing = list(age ~ "always", grade ~ "no")` or `missing = everything() ~ "no"`). The default is `everything() ~ "ifany"`, which adds a missing row only for variables that have missing values.
- `missing_text`: string indicating text shown on missing row. Default is `"Unknown"`.
- `missing_stat`: statistic to show on missing row. Default is `"{N_miss}"`. Possible values are `N_miss`, `N_obs`, `N_nonmiss`, `p_miss`, `p_nonmiss`.

sort (formula-list-selector)
: Specifies sorting to perform for categorical variables. Values must be one of `c("alphanumeric", "frequency")`. Default is `all_categorical(FALSE) ~ "alphanumeric"`.

col_label
: Used to override default labels in summary table, e.g. `list(age = "Age, years")`. The default for each variable is the column label attribute, `attr(., 'label')`. If no label has been set, the column name is used.

pre_output
: (`shiny.tag`) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output
: (`shiny.tag`) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like `shiny::helpText()` are useful.

transformators
: (`list` of `teal_transform_module`) that will be applied to transform module's data input. To learn more check `vignette("transform-input-data", package = "teal")`.

decorators
: **[Experimental]** (named `list` of lists of `teal_transform_module`) optional, decorator for tables or plots included in the module output reported. The decorators are applied to the respective output objects.

Value

Object of class `teal_module` to be used in `teal` applications.

Decorating Module

This module generates the following objects, which can be modified in place using decorators:

- `table` (`gtsummary` - output of `gtsummary::tbl_summary()`)

A Decorator is applied to the specific output using a named list of `teal_transform_module` objects. The name of this list corresponds to the name of the output to which the decorator is applied. See code snippet below:

```
tm_tbl_summary(
   ..., # arguments for module
   decorators = list(
     table = teal_transform_module(...) # applied to the `table` output
   )
)
```

For additional details and examples of decorators, refer to the vignette `vignette("decorate-module-output", package = "teal.modules.general")`.

To learn more please refer to the vignette `vignette("transform-module-output", package = "teal")` or the `teal::teal_transform_module()` documentation.

statistic argument

The statistic argument specifies the statistics presented in the table. The input dictates the summary statistics presented in the table. For example, `statistic = list(age ~ "{mean} ({sd})")` would report the mean and standard deviation for age; `statistic = list(all_continuous() ~ "{mean} ({sd})")` would report the mean and standard deviation for all continuous variables.

The values are interpreted using `glue::glue()` syntax: a name that appears between curly brackets will be interpreted as a function name and the formatted result of that function will be placed in the table.

For categorical variables, the following statistics are available to display: `{n}` (frequency), `{N}` (denominator), `{p}` (percent).

For continuous variables, **any univariate function may be used**. The most commonly used functions are `{median}`, `{mean}`, `{sd}`, `{min}`, and `{max}`. Additionally, `{p##}` is available for percentiles, where ## is an integer from 0 to 100. For example, `p25: quantile(probs=0.25, type=2)`.

When the summary type is `"continuous2"`, pass a vector of statistics. Each element of the vector will result in a separate row in the summary table.

For both categorical and continuous variables, statistics on the number of missing and non-missing observations and their proportions are available to display.

- `{N_obs}` total number of observations
- `{N_miss}` number of missing observations
- `{N_nonmiss}` number of non-missing observations
- `{p_miss}` percentage of observations missing
- `{p_nonmiss}` percentage of observations not missing

digits argument

The digits argument specifies the number of digits (or formatting function) statistics are rounded to.

The values passed can either be a single integer, a vector of integers, a function, or a list of functions. If a single integer or function is passed, it is recycled to the length of the number of statistics presented. For example, if the statistic is `"{mean} ({sd})"`, it is equivalent to pass `1`, `c(1, 1)`, `label_style_number(digits=1)`, and `list(label_style_number(digits=1), label_style_number(digits=1))`.

Named lists are also accepted to change the default formatting for a single statistic, e.g. `list(sd = label_style_number(digits=1))`.

type and value arguments

There are four summary types. Use the `type` argument to change the default summary types.

- `"continuous"` summaries are shown on a *single row*. Most numeric variables default to summary type continuous.
- `"continuous2"` summaries are shown on *2 or more rows*
- `"categorical"` *multi-line* summaries of nominal data. Character variables, factor variables, and numeric variables with fewer than 10 unique levels default to type categorical. To change a numeric variable to continuous that defaulted to categorical, use `type = list(varname ~ "continuous")`

- "dichotomous" categorical variables that are displayed on a *single row*, rather than one row per level of the variable. Variables coded as TRUE/FALSE, 0/1, or yes/no are assumed to be dichotomous, and the TRUE, 1, and yes rows are displayed. Otherwise, the value to display must be specified in the value argument, e.g. value = list(varname ~ "level to show")

Reporting

This module returns an object of class teal_module, that contains a server function. Since the server function returns a teal_report object, this makes this module reportable, which means that the reporting functionality will be turned on automatically by the teal framework.

For more information on reporting in teal, see the vignettes:

- vignette("reportable-shiny-application", package = "teal.reporter")
- vignette("adding-support-for-reporting-to-custom-modules", package = "teal")

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# General example
data <- teal_data()
data <- within(data, CO2 <- CO2)
app <- init(
  data = data,
  modules = modules(
    tm_tbl_summary(
      by = teal.picks::picks(
        datasets("CO2", "CO2"),
        variables(selected = "Plant")
      ),
      include = teal.picks::picks(
        datasets("CO2", "CO2"),
        variables(selected = c("Type", "Treatment"), multiple = TRUE)
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- within(teal.data::teal_data(), {
  ADSL <- teal.data::rADSL
  ADTTE <- teal.data::rADTTE
})
join_keys(data) <- default_cdisc_join_keys[names(data)]
app <- init(
```

```
    data = data,
    modules = modules(
      tm_tbl_summary(
        by = teal.picks::picks(
          datasets(c("ADSL", "ADTTE"), "ADTTE"),
          variables(c("SEX", "COUNTRY", "SITEID", "ACTARM", "CNSR", "PARAMCD"), "SEX")
        ),
        include = teal.picks::picks(
          datasets(c("ADSL", "ADTTE"), "ADSL"),
          variables(c("SITEID", "COUNTRY", "ACTARM", "SEX"), "SITEID", multiple = TRUE)
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }
```

`tm_variable_browser` `teal` *module: Variable browser*

Description

Module provides provides a detailed summary and visualization of variable distributions for `data.frame` objects, with interactive features to customize analysis.

Usage

```
tm_variable_browser(
  label = "Variable Browser",
  datasets_selected = deprecated(),
  datanames = if (missing(datasets_selected)) "all" else datasets_selected,
  parent_dataname = "ADSL",
  pre_output = NULL,
  post_output = NULL,
  ggplot2_args = teal.widgets::ggplot2_args(),
  transformators = list()
)
```

Arguments

`label` (`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.

`datasets_selected` (`character`) **[Deprecated]** vector of datasets to show, please use the `datanames` argument.

`datanames` (`character`) Names of the datasets relevant to the item. There are 2 reserved values that have specific behaviors:

- The keyword "all" includes all datasets available in the data passed to the teal application.
- NULL hides the sidebar panel completely.
- If transformators are specified, their datanames are automatically added to this datanames argument.

parent_dataname
: (character(1)) string specifying a parent dataset. If it exists in datanames then an extra checkbox will be shown to allow users to not show variables in other datasets which exist in this dataname. This is typically used to remove ADSL columns in CDISC data. In non CDISC data this can be ignored. Defaults to "ADSL".

pre_output
: (shiny.tag) optional, text or UI element to be displayed before the module's output, providing context or a title. with text placed before the output to put the output into context. For example a title.

post_output
: (shiny.tag) optional, text or UI element to be displayed after the module's output, adding context or further instructions. Elements like shiny::helpText() are useful.

ggplot2_args
: (ggplot2_args) object created by teal.widgets::ggplot2_args() with settings for the module plot. The argument is merged with options variable teal.ggplot2_args and default module setup.

  For more details see the vignette: vignette("custom-ggplot2-arguments", package = "teal.widgets")

transformators
: (list of teal_transform_module) that will be applied to transform module's data input. To learn more check vignette("transform-input-data", package = "teal").

Details

Numeric columns with fewer than 30 distinct values can be treated as either discrete or continuous with a checkbox allowing users to switch how they are treated(if < 6 unique values then the default is discrete, otherwise it is continuous).

Value

Object of class teal_module to be used in teal applications.

Examples in Shinylive

example-1 Open in Shinylive

example-2 Open in Shinylive

Examples

```
# general data example
data <- teal_data()
data <- within(data, {
  iris <- iris
  mtcars <- mtcars
```

```
  women <- women
  faithful <- faithful
  CO2 <- CO2
})

app <- init(
  data = data,
  modules = modules(
    tm_variable_browser(
      label = "Variable browser"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC example data
library(sparkline)
data <- teal_data()
data <- within(data, {
  ADSL <- teal.data::rADSL
  ADTTE <- teal.data::rADTTE
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_variable_browser(
      label = "Variable browser"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

Index

add_facet_labels, 3

dplyr::dplyr_tidy_select, *59*
DT::datatable(), *13*
DT::renderDataTable(), *13*

ggplot2::ggplotGrob(), *19*, *45*
ggrepel::geom_text_repel(), *10*
glue::glue(), *61*
gtsummary::tbl_summary, *59*
gtsummary::tbl_summary(), *60*

shiny::helpText(), *27*

teal.picks::picks(), *8, 9*, *22, 23*
teal.picks::variables(), *59*
teal.widgets::basic_table_args(), *55*
teal.widgets::ggplot2_args(), *5*, *9*, *19*, *23*, *27*, *31*, *36*, *45*, *48*, *64*
teal::teal_transform_module(), *6*, *10*, *20*, *24*, *28*, *32*, *36*, *41*, *46*, *49*, *52*, *56*, *60*
tm_a_pca, 4
tm_a_regression, 8
tm_data_table, 12
tm_file_viewer, 15
tm_front_page, 16
tm_g_association, 18
tm_g_bivariate, 22
tm_g_distribution, 26
tm_g_response, 30
tm_g_scatterplot, 34
tm_g_scatterplotmatrix, 39
tm_missing_data, 44
tm_outliers, 47
tm_rmarkdown, 51
tm_t_crosstable, 54
tm_tbl_summary, 58
tm_variable_browser, 63