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

October 7, 2024

Type Package

Title General Modules for 'teal' Applications

Version 0.3.0.9051

Date 2024-10-07

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 ggmosaic (>= 0.3.0), ggplot2 (>= 3.4.0), R (>= 3.6), shiny (>=
1.6.0), teal (>= 0.15.2.9052), teal.transform (>= 0.5.0)

Imports checkmate (>= 2.1.0), dplyr (>= 1.0.5), DT (>= 0.13), forcats
(>= 1.0.0), grid, scales, shinyjs, shinyTree (>= 0.2.8),
shinyvalidate, shinyWidgets (>= 0.5.1), stats, stringr (>=
1.4.1), teal.code (>= 0.5.0), teal.data (>= 0.5.0), teal.logger
(>= 0.2.0.9004), teal.reporter (>= 0.3.0), teal.widgets (>=
0.4.0), tern (>= 0.9.5), tibble (>= 2.0.0), tidyr (>= 0.8.3),
tools, utils

Suggests broom (>= 0.7.10), colourpicker, ggExtra, ggpmisc (>= 0.4.3),
ggpp, ggrepel, goftest, gridExtra, htmlwidgets, jsonlite, knitr
(>= 1.42), lattice (>= 0.18-4), logger (>= 0.2.0), MASS,
nestcolor (>= 0.1.0), pkgload, rlang (>= 1.0.0), rmarkdown (>=
2.23), roxy.shinylive, rtables (>= 0.6.8), rvest, shinytest2,
sparkline, testthat (>= 3.1.9), withr (>= 2.0.0)

VignetteBuilder knitr

Config/Needs/verdepcheck haleyjeppson/ggmosaic, tidyverse/ggplot2,
rstudio/shiny, insightsengineering/teal,

insightsengineering/teal.transform, mllg/checkmate,
tidyverse/dplyr, rstudio/DT, tidyverse/forcats, 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, tidymodels/broom,
daattali/colourpicker, daattali/ggExtra, aphalo/ggpmisc,
aphalo/ggpp, slowkow/ggrepel, baddstats/goftest,
ramnathv/htmlwidgets, jeroen/jsonlite, yihui/knitr,
daroczig/logger, deepayan/lattice,
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

Encoding UTF-8

Language en-US

LazyData true

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

RoxygenNote 7.3.2

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

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

RemoteRef HEAD

RemoteSha 643644eb0d839c755cc796f6074b6f8c4b9b59ea

Contents

Index **52**

---

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))
```

---

get_scatterplotmatrix_stats

Get stats for x-y pairs in scatterplot matrix

Description

Uses `stats::cor.test()` per default for all numerical input variables and converts results to character vector. Could be extended if different stats for different variable types are needed. Meant to be called from `lattice::panel.text()`.

Usage

```
get_scatterplotmatrix_stats(
  x,
  y,
  .f = stats::cor.test,
  .f_args = list(),
  round_stat = 2,
  round_pval = 4
)
```

Arguments

x, y	(`numeric`) vectors of data values. x and y must have the same length.
.f	(`function`) function that accepts x and y as formula input `~ x + y`. Default `stats::cor.test`.
.f_args	(`list`) of arguments to be passed to `.f`.
round_stat	(`integer(1)`) optional, number of decimal places to use when rounding the estimate.
round_pval	(`integer(1)`) optional, number of decimal places to use when rounding the p-value.

Details

Presently we need to use a formula input for `stats::cor.test` because `na.fail` only gets evaluated when a formula is passed (see below).

```
x = c(1,3,5,7,NA)
y = c(3,6,7,8,1)
stats::cor.test(x, y, na.action = "na.fail")
stats::cor.test(~ x + y,  na.action = "na.fail")
```

Value

Character with stats. For `stats::cor.test()` correlation coefficient and p-value.

Examples

```
set.seed(1)
x <- runif(25, 0, 1)
y <- runif(25, 0, 1)
x[c(3, 10, 18)] <- NA

get_scatterplotmatrix_stats(x, y, .f = stats::cor.test, .f_args = list(method = "pearson"))
get_scatterplotmatrix_stats(x, y, .f = stats::cor.test, .f_args = list(
  method = "pearson",
  na.action = na.fail
))
```

---

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,
  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
)
```

Arguments

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

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)
  USArrests <- USArrests
})

datanames(data) <- "USArrests"

app <- init(
  data = data,
  modules = modules(
    tm_a_pca(
      "PCA",
      dat = data_extract_spec(
        dataname = "USArrests",
        select = select_spec(
          choices = variable_choices(
            data = data[["USArrests"]], c("Murder", "Assault", "UrbanPop", "Rape")
          ),
          selected = c("Murder", "Assault"),
          multiple = TRUE
        ),
        filter = NULL
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
})
datanames(data) <- "ADSL"
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

app <- init(
  data = data,
  modules = modules(
    tm_a_pca(
      "PCA",
```

```
      dat = data_extract_spec(
        dataname = "ADSL",
        select = select_spec(
          choices = variable_choices(
            data = data[["ADSL"]], c("BMRKR1", "AGE", "EOSDY")
          ),
          selected = c("BMRKR1", "AGE"),
          multiple = TRUE
        ),
        filter = NULL
      )
    )
  )
)
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,
  response,
  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)
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
regressor	(data_extract_spec or list of multiple data_extract_spec) Regressor variables from an incoming dataset with filtering and selecting.
response	(data_extract_spec or list of multiple data_extract_spec) Response variables from an incoming dataset with filtering and selecting.
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 Fi
	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.

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

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
})
datanames(data) <- c("CO2")

app <- init(
  data = data,
  modules = modules(
    tm_a_regression(
      label = "Regression",
      response = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = "uptake",
```

```
                    selected = "uptake",
                    multiple = FALSE,
                    fixed = TRUE
                  )
                ),
                regressor = data_extract_spec(
                  dataname = "CO2",
                  select = select_spec(
                    label = "Select variables:",
                    choices = variable_choices(data[["CO2"]], c("conc", "Treatment")),
                    selected = "conc",
                    multiple = TRUE,
                    fixed = FALSE
                  )
                )
              )
            )
          )
          if (interactive()) {
            shinyApp(app$ui, app$server)
          }

          # CDISC data example
          data <- teal_data()
          data <- within(data, {
            require(nestcolor)
            ADSL <- rADSL
          })
          datanames(data) <- "ADSL"
          join_keys(data) <- default_cdisc_join_keys[datanames(data)]

          app <- init(
            data = data,
            modules = modules(
              tm_a_regression(
                label = "Regression",
                response = data_extract_spec(
                  dataname = "ADSL",
                  select = select_spec(
                    label = "Select variable:",
                    choices = "BMRKR1",
                    selected = "BMRKR1",
                    multiple = FALSE,
                    fixed = TRUE
                  )
                ),
                regressor = data_extract_spec(
                  dataname = "ADSL",
                  select = select_spec(
                    label = "Select variables:",
                    choices = variable_choices(data[["ADSL"]], c("AGE", "SEX", "RACE")),
                    selected = "AGE",
                    multiple = TRUE,
```

```
        fixed = FALSE
      )
    )
  )
)
)
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 = character(0),
  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
)
```

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) A vector of datasets which should be shown and in what order.
                    Names in the vector have to correspond with datasets names. If vector of length
                    == 0 (default) then all datasets are shown. Note: Only datasets of the data.frame
                    class are compatible.

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.

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
})
datanames(data) <- c("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 <- rADSL
})
datanames(data) <- "ADSL"
join_keys(data) <- default_cdisc_join_keys[datanames(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)
})
datanames(data) <- c("data")

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 = "https://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 = FALSE
)
```

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) indicating whether the metadata of the datasets be available on the
                   module.

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 <- rADSL
  attr(ADSL, "metadata") <- list("Author" = "NEST team", "data_source" = "synthetic data")
})
datanames(data) <- "ADSL"
join_keys(data) <- default_cdisc_join_keys[datanames(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"),
          show_metadata = TRUE
        )
      ),
      header = tags$h1("Sample Application"),
      footer = tags$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,
  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()
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
ref	(data_extract_spec or list of multiple data_extract_spec) Reference variable, must accepts a data_extract_spec with select_spec(multiple = FALSE) to ensure single selection option.
vars	(data_extract_spec or list of multiple data_extract_spec) Variables to be associated with the reference variable.
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").

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

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)
})
datanames(data) <- c("CO2")

app <- init(
  data = data,
  modules = modules(
    tm_g_association(
      ref = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["CO2"]], c("Plant", "Type", "Treatment")),
          selected = "Plant",
          fixed = FALSE
        )
      ),
      vars = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variables:",
          choices = variable_choices(data[["CO2"]], c("Plant", "Type", "Treatment")),
          selected = "Treatment",
          multiple = TRUE,
          fixed = FALSE
        )
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
})
datanames(data) <- "ADSL"
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

app <- init(
  data = data,
```

```
        modules = modules(
          tm_g_association(
            ref = data_extract_spec(
              dataname = "ADSL",
              select = select_spec(
                label = "Select variable:",
                choices = variable_choices(
                  data[["ADSL"]],
                  c("SEX", "RACE", "COUNTRY", "ARM", "STRATA1", "STRATA2", "ITTFL", "BMRKR2")
                ),
                selected = "RACE",
                fixed = FALSE
              )
            ),
            vars = data_extract_spec(
              dataname = "ADSL",
              select = select_spec(
                label = "Select variables:",
                choices = variable_choices(
                  data[["ADSL"]],
                  c("SEX", "RACE", "COUNTRY", "ARM", "STRATA1", "STRATA2", "ITTFL", "BMRKR2")
                ),
                selected = "BMRKR2",
                multiple = TRUE,
                fixed = FALSE
              )
            )
          )
        )
      )
      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,
  y,
  row_facet = NULL,
```

```
    col_facet = NULL,
    facet = !is.null(row_facet) || !is.null(col_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
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
x	(data_extract_spec or list of multiple data_extract_spec) Variable names selected to plot along the x-axis by default. Can be numeric, factor or character. No empty selections are allowed.
y	(data_extract_spec or list of multiple data_extract_spec) Variable names selected to plot along the y-axis by default. Can be numeric, factor or character.
row_facet	(data_extract_spec or list of multiple data_extract_spec) optional, specification of the data variable(s) to use for faceting rows.
col_facet	(data_extract_spec or list of multiple data_extract_spec) optional, specification of the data variable(s) to use for faceting columns.
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	(data_extract_spec or list of multiple data_extract_spec) 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.
fill	(data_extract_spec or list of multiple data_extract_spec) 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.
size	(data_extract_spec or list of multiple data_extract_spec) 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.
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 of-
                fered 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 set-
                tings 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 out-
                put, adding context or further instructions. Elements like shiny::helpText()
                are useful.

Details

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

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](#)

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)
})
datanames(data) <- c("CO2")

app <- init(
  data = data,
  modules = tm_g_bivariate(
    x = data_extract_spec(
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]]),
        selected = "conc",
        fixed = FALSE
      )
    ),
    y = data_extract_spec(
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]]),
        selected = "uptake",
        multiple = FALSE,
        fixed = FALSE
      )
    ),
    row_facet = data_extract_spec(
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]]),
        selected = "Type",
        fixed = FALSE
      )
    ),
    col_facet = data_extract_spec(
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]]),
        selected = "Treatment",
        fixed = FALSE
      )
    )
  )
)
if (interactive()) {
```

```
    shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
})
datanames(data) <- c("ADSL")
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

app <- init(
  data = data,
  modules = tm_g_bivariate(
    x = data_extract_spec(
      dataname = "ADSL",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["ADSL"]]),
        selected = "AGE",
        fixed = FALSE
      )
    ),
    y = data_extract_spec(
      dataname = "ADSL",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["ADSL"]]),
        selected = "SEX",
        multiple = FALSE,
        fixed = FALSE
      )
    ),
    row_facet = data_extract_spec(
      dataname = "ADSL",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["ADSL"]]),
        selected = "ARM",
        fixed = FALSE
      )
    ),
    col_facet = data_extract_spec(
      dataname = "ADSL",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["ADSL"]]),
        selected = "COUNTRY",
        fixed = FALSE
      )
    )
  )
```

```
  )
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,
  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
)
```

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	(data_extract_spec or list of multiple data_extract_spec) Variable(s) for which the distribution will be analyzed.
strata_var	(data_extract_spec or list of multiple data_extract_spec) Categorical variable used to split the distribution analysis.
group_var	(data_extract_spec or list of multiple data_extract_spec) 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.

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
})
datanames(data) <- "iris"

app <- init(
  data = data,
  modules = list(
    tm_g_distribution(
      dist_var = data_extract_spec(
        dataname = "iris",
```

```
            select = select_spec(variable_choices("iris"), "Petal.Length")
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }


  # CDISC data example
  data <- teal_data()
  data <- within(data, {
    ADSL <- rADSL
  })
  datanames(data) <- c("ADSL")
  join_keys(data) <- default_cdisc_join_keys[datanames(data)]

  vars1 <- choices_selected(
    variable_choices(data[["ADSL"]], c("ARM", "COUNTRY", "SEX")),
    selected = NULL
  )

  app <- init(
    data = data,
    modules = modules(
      tm_g_distribution(
        dist_var = data_extract_spec(
          dataname = "ADSL",
          select = select_spec(
            choices = variable_choices(data[["ADSL"]], c("AGE", "BMRKR1")),
            selected = "BMRKR1",
            multiple = FALSE,
            fixed = FALSE
          )
        ),
        strata_var = data_extract_spec(
          dataname = "ADSL",
          filter = filter_spec(
            vars = vars1,
            multiple = TRUE
          )
        ),
        group_var = data_extract_spec(
          dataname = "ADSL",
          filter = filter_spec(
            vars = vars1,
            multiple = TRUE
          )
        )
      )
    )
  )
```

```
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,
  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
)
```

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For modules() defaults to "root". See Details.
response	(data_extract_spec or list of multiple data_extract_spec) Which variable to use as the response. You can define one fixed column by setting fixed = TRUE inside the select_spec.
	The data_extract_spec must not allow multiple selection in this case.
x	(data_extract_spec or list of multiple data_extract_spec) Specifies which variable to use on the X-axis of the response plot. Allow the user to select multiple columns from the data allowed in teal.
	The data_extract_spec must not allow multiple selection in this case.

row_facet	(data_extract_spec or list of multiple data_extract_spec) optional specification of the data variable(s) to use for faceting rows.
col_facet	(data_extract_spec or list of multiple data_extract_spec) 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.

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](#)

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]])
  }
})
datanames(data) <- "mtcars"

app <- init(
  data = data,
  modules = modules(
    tm_g_response(
      label = "Response Plots",
      response = data_extract_spec(
        dataname = "mtcars",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["mtcars"]], c("cyl", "gear")),
          selected = "cyl",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      x = data_extract_spec(
        dataname = "mtcars",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["mtcars"]], c("vs", "am")),
          selected = "vs",
          multiple = FALSE,
          fixed = FALSE
        )
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
})
datanames(data) <- c("ADSL")
join_keys(data) <- default_cdisc_join_keys[datanames(data)]
```

```
app <- init(
  data = data,
  modules = modules(
    tm_g_response(
      label = "Response Plots",
      response = data_extract_spec(
        dataname = "ADSL",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["ADSL"]], c("BMRKR2", "COUNTRY")),
          selected = "BMRKR2",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      x = data_extract_spec(
        dataname = "ADSL",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["ADSL"]], c("SEX", "RACE")),
          selected = "RACE",
          multiple = FALSE,
          fixed = FALSE
        )
      )
    )
  )
)
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 trans-
formations, trend line additions, and dynamic adjustments of point opacity and size through UI
controls.

Usage

```
tm_g_scatterplot(
  label = "Scatterplot",
  x,
  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()
)
```

Arguments

`label`	(`character(1)`) Label shown in the navigation item for the module or module group. For `modules()` defaults to `"root"`. See `Details`.
`x`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) Specifies variable names selected to plot along the x-axis by default.
`y`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) Specifies variable names selected to plot along the y-axis by default.
`color_by`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) optional, defines the color encoding. If `NULL` then no color encoding option will be displayed.
`size_by`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) optional, defines the point size encoding. If `NULL` then no size encoding option will be displayed.
`row_facet`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) optional, specifies the variable(s) for faceting rows.
`col_facet`	(`data_extract_spec` or `list` of multiple `data_extract_spec`) 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")

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

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
})
datanames(data) <- "CO2"

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplot(
      label = "Scatterplot Choices",
      x = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["CO2"]], c("conc", "uptake")),
          selected = "conc",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      y = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["CO2"]], c("conc", "uptake")),
          selected = "uptake",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      color_by = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(
            data[["CO2"]],
            c("Plant", "Type", "Treatment", "conc", "uptake")
          ),
          selected = NULL,
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      size_by = data_extract_spec(
        dataname = "CO2",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["CO2"]], c("conc", "uptake")),
          selected = "uptake",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      row_facet = data_extract_spec(
```

```
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]], c("Plant", "Type", "Treatment")),
        selected = NULL,
        multiple = FALSE,
        fixed = FALSE
      )
    ),
    col_facet = data_extract_spec(
      dataname = "CO2",
      select = select_spec(
        label = "Select variable:",
        choices = variable_choices(data[["CO2"]], c("Plant", "Type", "Treatment")),
        selected = NULL,
        multiple = FALSE,
        fixed = FALSE
      )
    )
  )
 )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}


# CDISC data example
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
})
datanames(data) <- c("ADSL")
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplot(
      label = "Scatterplot Choices",
      x = data_extract_spec(
        dataname = "ADSL",
        select = select_spec(
          label = "Select variable:",
          choices = variable_choices(data[["ADSL"]], c("AGE", "BMRKR1", "BMRKR2")),
          selected = "AGE",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      y = data_extract_spec(
        dataname = "ADSL",
```

```
    select = select_spec(
      label = "Select variable:",
      choices = variable_choices(data[["ADSL"]], c("AGE", "BMRKR1", "BMRKR2")),
      selected = "BMRKR1",
      multiple = FALSE,
      fixed = FALSE
    )
  ),
  color_by = data_extract_spec(
    dataname = "ADSL",
    select = select_spec(
      label = "Select variable:",
      choices = variable_choices(
        data[["ADSL"]],
        c("AGE", "BMRKR1", "BMRKR2", "RACE", "REGION1")
      ),
      selected = NULL,
      multiple = FALSE,
      fixed = FALSE
    )
  ),
  size_by = data_extract_spec(
    dataname = "ADSL",
    select = select_spec(
      label = "Select variable:",
      choices = variable_choices(data[["ADSL"]], c("AGE", "BMRKR1")),
      selected = "AGE",
      multiple = FALSE,
      fixed = FALSE
    )
  ),
  row_facet = data_extract_spec(
    dataname = "ADSL",
    select = select_spec(
      label = "Select variable:",
      choices = variable_choices(data[["ADSL"]], c("BMRKR2", "RACE", "REGION1")),
      selected = NULL,
      multiple = FALSE,
      fixed = FALSE
    )
  ),
  col_facet = data_extract_spec(
    dataname = "ADSL",
    select = select_spec(
      label = "Select variable:",
      choices = variable_choices(data[["ADSL"]], c("BMRKR2", "RACE", "REGION1")),
      selected = NULL,
      multiple = FALSE,
      fixed = FALSE
    )
  )
 )
)
```

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

Arguments

label	(character(1)) Label shown in the navigation item for the module or module group. For `modules()` defaults to "root". See `Details`.
variables	(data_extract_spec or list of multiple data_extract_spec) Specifies plotting variables from an incoming dataset with filtering and selecting. In case of `data_extract_spec` use `select_spec(..., ordered = TRUE)` if plot elements should be rendered according to selection order.
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.

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

Note

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

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)
  )
})
datanames(data) <- c("countries", "sales")
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(
          data_extract_spec(
            dataname = "countries",
            select = select_spec(
              label = "Select variables:",
              choices = variable_choices(data[["countries"]]),
              selected = c("area", "gdp", "debt"),
              multiple = TRUE,
              ordered = TRUE,
              fixed = FALSE
            )
          ),
          data_extract_spec(
            dataname = "sales",
            filter = filter_spec(
              label = "Select variable:",
              vars = "country_id",
              choices = value_choices(data[["sales"]], "country_id"),
              selected = c("DE", "FR", "IT", "ES", "PT", "GR", "NL", "BE", "LU", "AT"),
              multiple = TRUE
            ),
            select = select_spec(
              label = "Select variables:",
             choices = variable_choices(data[["sales"]], c("quantity", "costs", "profit")),
              selected = c("quantity", "costs", "profit"),
              multiple = TRUE,
              ordered = TRUE,
              fixed = FALSE
            )
          )
        )
      )
    )
  )
  if (interactive()) {
    shinyApp(app$ui, app$server)
  }


  # CDISC data example
  data <- teal_data()
  data <- within(data, {
    ADSL <- rADSL
    ADRS <- rADRS
```

```
})
datanames(data) <- c("ADSL", "ADRS")
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplotmatrix(
      label = "Scatterplot matrix",
      variables = list(
        data_extract_spec(
          dataname = "ADSL",
          select = select_spec(
            label = "Select variables:",
            choices = variable_choices(data[["ADSL"]]),
            selected = c("AGE", "RACE", "SEX"),
            multiple = TRUE,
            ordered = TRUE,
            fixed = FALSE
          )
        ),
        data_extract_spec(
          dataname = "ADRS",
          filter = filter_spec(
            label = "Select endpoints:",
            vars = c("PARAMCD", "AVISIT"),
          choices = value_choices(data[["ADRS"]], c("PARAMCD", "AVISIT"), c("PARAM", "AVISIT")),
            selected = "INVET - END OF INDUCTION",
            multiple = TRUE
          ),
          select = select_spec(
            label = "Select variables:",
            choices = variable_choices(data[["ADRS"]]),
            selected = c("AGE", "AVAL", "ADY"),
            multiple = TRUE,
            ordered = TRUE,
            fixed = FALSE
          )
        )
      )
    )
  )
)
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,
  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
)
```

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.
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 out-
               put, adding context or further instructions. Elements like shiny::helpText()
               are useful.

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 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"]])
})
datanames(data) <- c("iris", "mtcars")

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


# CDISC example data
data <- teal_data()
data <- within(data, {
  require(nestcolor)
  ADSL <- rADSL
  ADRS <- rADRS
```

```
  })
  datanames(data) <- c("ADSL", "ADRS")
  join_keys(data) <- default_cdisc_join_keys[datanames(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,
  categorical_var = NULL,
 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
)
```

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	(data_extract_spec or list of multiple data_extract_spec) Specifies variable(s) to be analyzed for outliers.
categorical_var	
	(data_extract_spec or list of multiple data_extract_spec) 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.

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, {
  CO2 <- CO2
  CO2[["primary_key"]] <- seq_len(nrow(CO2))
})
datanames(data) <- "CO2"
join_keys(data) <- join_keys(join_key("CO2", "CO2", "primary_key"))

vars <- choices_selected(variable_choices(data[["CO2"]], c("Plant", "Type", "Treatment")))

app <- init(
  data = data,
  modules = modules(
    tm_outliers(
      outlier_var = list(
        data_extract_spec(
          dataname = "CO2",
          select = select_spec(
```

```
              label = "Select variable:",
              choices = variable_choices(data[["CO2"]], c("conc", "uptake")),
              selected = "uptake",
              multiple = FALSE,
              fixed = FALSE
            )
          )
        ),
        categorical_var = list(
          data_extract_spec(
            dataname = "CO2",
            filter = filter_spec(
              vars = vars,
              choices = value_choices(data[["CO2"]], vars$selected),
              selected = value_choices(data[["CO2"]], vars$selected),
              multiple = TRUE
            )
          )
        )
      )
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# CDISC data example
data <- teal_data()
data <- within(data, {
  ADSL <- rADSL
})
datanames(data) <- "ADSL"
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

fact_vars_adsl <- names(Filter(isTRUE, sapply(data[["ADSL"]], is.factor)))
vars <- choices_selected(variable_choices(data[["ADSL"]], fact_vars_adsl))

app <- init(
  data = data,
  modules = modules(
    tm_outliers(
      outlier_var = list(
        data_extract_spec(
          dataname = "ADSL",
          select = select_spec(
            label = "Select variable:",
            choices = variable_choices(data[["ADSL"]], c("AGE", "BMRKR1")),
            selected = "AGE",
            multiple = FALSE,
            fixed = FALSE
          )
        )
      ),
```

```
        categorical_var = list(
          data_extract_spec(
            dataname = "ADSL",
            filter = filter_spec(
              vars = vars,
              choices = value_choices(data[["ADSL"]], vars$selected),
              selected = value_choices(data[["ADSL"]], vars$selected),
              multiple = TRUE
            )
          )
        )
      )
    )
  )
)
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,
  y,
  show_percentage = TRUE,
  show_total = TRUE,
  pre_output = NULL,
  post_output = NULL,
  basic_table_args = teal.widgets::basic_table_args()
)
```

Arguments

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

x   (data_extract_spec or list of multiple data_extract_spec) Object with
                 all available choices with pre-selected option for variable X - row values. In
                 case of data_extract_spec use select_spec(..., ordered = TRUE) if table
                 elements should be rendered according to selection order.

y	(data_extract_spec or list of multiple data_extract_spec) Object with all available choices with pre-selected option for variable Y - column values. data_extract_spec 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.
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")

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

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))
})
datanames(data) <- "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 = data_extract_spec(
            dataname = "mtcars",
            select = select_spec(
              label = "Select variable:",
              choices = variable_choices(data[["mtcars"]], c("cyl", "vs", "am", "gear")),
              selected = c("cyl", "gear"),
              multiple = TRUE,
              ordered = TRUE,
              fixed = FALSE
            )
          ),
          y = data_extract_spec(
            dataname = "mtcars",
            select = select_spec(
              label = "Select variable:",
              choices = variable_choices(data[["mtcars"]], c("cyl", "vs", "am", "gear")),
              selected = "vs",
              multiple = FALSE,
              fixed = FALSE
            )
          )
        )
      )
    )
    if (interactive()) {
      shinyApp(app$ui, app$server)
    }


    # CDISC data example
    data <- teal_data()
    data <- within(data, {
      ADSL <- rADSL
    })
    datanames(data) <- "ADSL"
    join_keys(data) <- default_cdisc_join_keys[datanames(data)]

    app <- init(
      data = data,
      modules = modules(
        tm_t_crosstable(
          label = "Cross Table",
          x = data_extract_spec(
            dataname = "ADSL",
            select = select_spec(
              label = "Select variable:",
              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
        )
      ),
      y = data_extract_spec(
        dataname = "ADSL",
        select = select_spec(
          label = "Select variable:",
          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
        )
      )
    )
  )
)
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 = character(0),
  parent_dataname = "ADSL",
  pre_output = NULL,
  post_output = NULL,
  ggplot2_args = teal.widgets::ggplot2_args()
)
```

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) vector of datasets which should be shown, in order. Names must
                     correspond with datasets names. If vector of length zero (default) then all
                     datasets are shown. Note: Only data.frame objects are compatible; using other
                     types will cause an error.

parent_dataname
                     (character(1)) string specifying a parent dataset. If it exists in datasets_selectedthen
                     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 out-
                     put, adding context or further instructions. Elements like shiny::helpText()
                     are useful.

ggplot2_args   (ggplot2_args) object created by [teal.widgets::ggplot2_args()](#) with set-
                     tings 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")

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
})
datanames(data) <- c("iris", "mtcars", "women", "faithful", "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 <- rADSL
  ADTTE <- rADTTE
})
datanames(data) <- c("ADSL", "ADTTE")
join_keys(data) <- default_cdisc_join_keys[datanames(data)]

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

Index