Package 'uteals'

October 14, 2026

Title Shared Utilities to Extend the 'teal' Modules

Language en-US

Version 0.0.6

Date 2026-10-14

URL https://github.com/phuse-org/uteals

Description Provides decorators, transformators, and utility functions to extend the 'teal' framework for interactive data analysis applications. Implements methods for data visualization enhancement, statistical data transformations, and workflow integration tools. Designed to support clinical and pharmaceutical research workflows within the 'teal' ecosystem through modular and reusable components.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports DT, R6, checkmate, cowplot, dplyr, formatters, ggplot2, ggplotify, gridify, jsonlite, junco, methods, openxlsx, patchwork, rlang, rtables, shiny (>= 1.11.0), shinyWidgets, shinyjs, teal, teal.code, teal.modules.clinical, teal.reporter, tern, yaml, zip

Suggests forcats, knitr, rAccess, rmarkdown, teal.data, teal.modules.general, teal.transform

VignetteBuilder knitr

Config/Needs/website insightsengineering/nesttemplate

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Nadia Abraham [aut], Chanchal Bhalla [aut], Peyman Eshghi [aut, cre], Pranith Gourisetty [aut], Sohan Lal [aut], Przemyslaw Posiadala [aut],

Alina Tselinina [aut],
Konrad Pagacz [aut],
PHUSE [cph]

Maintainer Peyman Eshghi <peymaan.es@gmail.com>

Repository CRAN

Date/Publication 2026-10-14 15:40:02 UTC

Contents

uteals-package 2
create_rel_risk_transformator 3
extract_modules_to_yaml 4
forestplot_x_decorator 5
ggplot_decorator 7
g_forest_with_transform 9
keep_by_label 11
merge_levels_transformator 12
or_filtering_transformator 13
patchwork_plot_decorator 14
remove_by_label 15
title_footer_decorator 15
tm_report_manager 17
use_freeze_tables 18
watermark_decorator 20

Index **21**

`uteals-package` *Decorators and transformators for* `teal` *modules.*

Description

Provides decorator and transformator modules for `teal` modules.

Author(s)

Maintainer: Peyman Eshghi <peymaan.es@gmail.com>

Authors:

- Peyman Eshghi <peymaan.es@gmail.com>
- Nadia Abraham <NAbraha5@ITS.JNJ.com>
- Chanchal Bhalla <CBhalla@ITS.JNJ.com>
- Pranith Gourisetty <PGourise@ITS.JNJ.com>
- Sohan Lal <SLal8@ITS.JNJ.com>
- Przemyslaw Posiadala <PPosiada@ITS.JNJ.com>

- Alina Tselinina <ATselini@ITS.JNJ.com>
- Konrad Pagacz <konrad.pagacz@gmail.com>

Other contributors:

- PHUSE [copyright holder]

See Also

Useful links:

- https://github.com/phuse-org/uteals

`create_rel_risk_transformator` *Create Relative risk column*

Description

[Experimental]

Usage

```
create_rel_risk_transformator(dataname, column_name, control_group, label_name)
```

Arguments

`dataname`	(`character(1)`) the name of the dataset which columns will be used for possible transformation. `dataname` should be passed in quotes ex: `"ADSL"`.
`column_name`	(`character(1)`) field or variable from the dataset.
`control_group`	(`character(1)`) one of the existing level from the selected `column_name`.
`label_name`	(`character(1)`) label for the new field or variable.

Details

This transformator allows the user to select a column and control group from the dataset and create a relative risk column.

Value

`teal::teal_transform_module`

Examples

```
app <- teal::init(
  data = teal.data::teal_data(IRIS = iris, code = "IRIS <- iris"),
  modules = teal::modules(
    teal::example_module(
      transformators = list(
        create_rel_risk_transformator("IRIS",
          label_name = "test",
          control_group = "setosa", column_name = "Species"
        )
      )
    )
  )
)
if (interactive()) {
  shiny::shinyApp(app$ui, app$server)
}
```

---

extract_modules_to_yaml

Extract Non-Parent Module Labels to YAML

Description

[Experimental] Extracts module labels from a teal modules object, filters out parent modules (grouping containers), and generates a YAML file with the functional modules.

Usage

```
extract_modules_to_yaml(mods, filepath, verbose = FALSE)
```

Arguments

mods	(teal_module or teal_modules) a teal modules object containing the module structure.
filepath	(character(1)) character string specifying the output YAML file path.
verbose	(logical(1)) whether to print informational messages. Default is FALSE.

Value

Character vector of non-parent module labels

Examples

```
# Extract modules from mods object to YAML file
mods <- teal::modules(
  teal::example_module("mod1"),
  teal::example_module("mod2")
)
labels <- extract_modules_to_yaml(mods, "panel_str_modules.yml")
unlink("panel_str_modules.yml")

# Clean up
if (file.exists("panel_str_modules.yml")) {
  file.remove("panel_str_modules.yml")
}
```

`forestplot_x_decorator` *Change X-axis scaling decorator for* `teal.modules.clinical::tm_g_forest_rsp`.

Description

[Experimental] A function to create a UI component for selecting a transform function for the forest plot x axis.

Usage

```
forestplot_x_decorator()
```

Details

The module creates a UI with a radio control for selecting the transform function. The selected transformation function is applied to the forest plot to update the plot's axis and annotations accordingly.

Value

`teal::teal_transform_module` Returns a modified plot object with the transformation applied.

Examples

```
library(teal.modules.clinical)
data <- teal.data::teal_data()
data <- within(data, {
  ADSL <- teal.modules.clinical::tmc_ex_adsl
  ADRS <- teal.modules.clinical::tmc_ex_adrs |>
    dplyr::mutate(AVALC = tern::d_onco_rsp_label(AVALC) |>
      formatters::with_label("Character Result/Finding")) |>
    dplyr::filter(PARAMCD != "OVRINV" | AVISIT == "FOLLOW UP")
```

```
})
teal.data::join_keys(data) <- teal.data::default_cdisc_join_keys[names(data)]

ADSL <- data[["ADSL"]]
ADRS <- data[["ADRS"]]

arm_ref_comp <- list(
  ARM = list(
    ref = "B: Placebo",
    comp = c("A: Drug X", "C: Combination")
  ),
  ARMCD = list(
    ref = "ARM B",
    comp = c("ARM A", "ARM C")
  )
)

app <- teal::init(
  data = data,
  modules = teal::modules(
    teal.modules.clinical::tm_g_forest_rsp(
      label = "Forest Response",
      dataname = "ADRS",
      arm_var = choices_selected(
        variable_choices(ADSL, c("ARM", "ARMCD")),
        "ARMCD"
      ),
      arm_ref_comp = arm_ref_comp,
      paramcd = choices_selected(
        value_choices(ADRS, "PARAMCD", "PARAM"),
        "INVET"
      ),
      subgroup_var = choices_selected(
        variable_choices(ADSL, names(ADSL)),
        c("BMRKR2", "SEX")
      ),
      strata_var = choices_selected(
        variable_choices(ADSL, c("STRATA1", "STRATA2")),
        "STRATA2"
      ),
      plot_height = c(600L, 200L, 2000L),
      decorators = list(
        plot = forestplot_x_decorator()
      ),
      default_responses = list(
        BESRSPI = list(
          rsp = c("Stable Disease (SD)", "Not Evaluable (NE)"),
          levels = c(
            "Complete Response (CR)", "Partial Response (PR)", "Stable Disease (SD)",
            "Progressive Disease (PD)", "Not Evaluable (NE)"
          )
        ),
        INVET = list(
```

```
            rsp = c("Complete Response (CR)", "Partial Response (PR)"),
            levels = c(
              "Complete Response (CR)", "Not Evaluable (NE)", "Partial Response (PR)",
              "Progressive Disease (PD)", "Stable Disease (SD)"
            )
          ),
          OVRINV = list(
            rsp = c("Progressive Disease (PD)", "Stable Disease (SD)"),
          levels = c("Progressive Disease (PD)", "Stable Disease (SD)", "Not Evaluable (NE)")
          )
        )
      )
    )
  )
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

ggplot_decorator ggplot *decorator*

Description

[Experimental] Decorator function to update various settings for ggplot plot objects

Usage

```
ggplot_decorator(
  output_name,
  label_text = "decorator",
  render_ui = c(),
  plot_options = list(title = "", footnote = "", y_breaks = "", y_limits_max = "",
    y_limits_min = "", x_breaks = "", x_labels_discrete = "", x_labels_cont = "",
     y_labels_discrete = "", y_labels_cont = "", font_size_geom_text = "",
   font_size_plot_title = "", font_size_axis_title = "", font_size_axis_text = "")
)
```

Arguments

output_name	a name for the output plot object.
label_text	customized label text for the decorator
render_ui	vector of ggplot_options from the following list: "title" - Title of the plot, "footnote" - Footnote of the plot, "y_breaks" - Value of breaks(numeric) for y-axis. Note: y_limits_max and y_limits_min should also be provided, "y_limits_max" - Value of y-axis maximum limit(numeric). Note: y_limits_max and y_breaks should also be provided, "y_limits_min" - Value of y-axis minimum limit(numeric). Note: y_breaks and y_limits_min should also be provided, "x_breaks"- Value

of breaks for continuous x-axis(numeric). Note: should be comma separated, "x_labels_discrete" - Values of labels for discrete x-axis. Note: should be comma separated, "x_labels_cont" - Values of labels for continuous x-axis. Note: should be comma separated, "y_labels_discrete" - Values of labels for discrete y-axis. Note: should be comma separated, "y_labels_cont" - Values of labels for continuous y-axis. Note: should be comma separated, "font_size_geom_text" - Font size of geom_text. Note: numeric value should be provided, "font_size_plot_title"- Font size of plot title text. Note: numeric value should be provided, "font_size_axis_title"- Font size of axis title text. Note: numeric value should be provided, "font_size_axis_text"- Font size of axis labels text. Note: numeric value should be provided

`plot_options` named list with the list of values for the `ggplot` options. The app developer can specify the required list of options while calling the decorator.

Details

The module creates a UI with text controls for specifying the list of `ggplot` options given in the `plot_options` parameter value. The entered `ggplot` options are applied to `ggplot` plot object.

Value

`teal::teal_transform_module` Returns a modified plot object with the transformation applied.

Examples

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

# teal.modules.general >= 0.6.0
app <- teal::init(
  data = data,
  modules = teal::modules(
    teal.modules.general::tm_g_scatterplot(
      label = "Scatterplot Choices",
      x = teal.transform::data_extract_spec(
        dataname = "ADSL",
        select = teal.transform::select_spec(
          label = "Select variable:",
       choices = teal.transform::variable_choices(data[["ADSL"]], c("AGE", "BMRKR1", "BMRKR2")),
          selected = "AGE",
          multiple = FALSE,
          fixed = FALSE
        )
      ),
      y = teal.transform::data_extract_spec(
        dataname = "ADSL",
        select = teal.transform::select_spec(
          label = "Select variable:",
       choices = teal.transform::variable_choices(data[["ADSL"]], c("AGE", "BMRKR1", "BMRKR2")),
```

```
            selected = "BMRKR1",
            multiple = FALSE,
            fixed = FALSE
          )
        ),
        decorators = list(
          plot = ggplot_decorator(
           output_name = "plot", render_ui = c("title", "footnote", "font_size_axis_title")
          )
        )
      )
    )
  )

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

---

g_forest_with_transform *Create a forest plot from an* rtable *with x transform settings*

Description

[Experimental] Given a rtables::rtable() object with at least one column with a single value and one column with 2 values, converts table to a ggplot2::ggplot() object and generates an accompanying forest plot. The table and forest plot are printed side-by-side.

Usage

```
g_forest_with_transform(
  tbl,
  col_x = attr(tbl, "col_x"),
  col_ci = attr(tbl, "col_ci"),
  vline = 1,
  forest_header = attr(tbl, "forest_header"),
  xlim = c(0.1, 10),
  transform_x = NA,
  x_at = c(0.1, 1, 10),
  width_columns = NULL,
  lbl_col_padding = 0,
  rel_width_forest = 0.25,
  font_size = 12,
  col_symbol_size = attr(tbl, "col_symbol_size"),
  col = getOption("ggplot2.discrete.colour")[1],
  ggtheme = NULL,
  as_list = FALSE
)
```

Arguments

tbl	(VTableTree) rtables table with at least one column with a single value and one column with 2 values.
col_x	(integer(1) or NULL) column index with estimator. By default tries to get this from tbl attribute col_x, otherwise needs to be manually specified. If NULL, points will be excluded from forest plot.
col_ci	(integer(1) or NULL) column index with confidence intervals. By default tries to get this from tbl attribute col_ci, otherwise needs to be manually specified. If NULL, lines will be excluded from forest plot.
vline	(numeric) position of the vertical reference line on the plot. like 0 and 1 in forest plot.
forest_header	(character(2)) text displayed to the left and right of vline, respectively. If vline = NULL then forest_header is not printed. By default tries to get this from tbl attribute forest_header. If NULL, defaults will be extracted from the table if possible, and set to "Comparison\nBetter" and "Treatment\nBetter" if not.
xlim	(numeric) range of x-axis limits. Example: c(0.1, 10)
transform_x	(character) function for x-values transformation
x_at	(numeric) specifies the tick marks on the axis. The value of union(xlim, vline) or Example: c(0.1,1,10).
width_columns	(numeric) a vector of column widths. Each element's position in colwidths corresponds to the column of tbl in the same position. If NULL, column widths are calculated according to maximum number of characters per column.
lbl_col_padding	(numeric) padding between label and columns value. Default is 0.
rel_width_forest	(proportion) proportion of total width to allocate to the forest plot. Relative width of table is then 1 - rel_width_forest. If as_list = TRUE, this parameter is ignored.
font_size	(numeric(1)) font size.
col_symbol_size	(numeric or NULL) column index from tbl containing data to be used to determine relative size for estimator plot symbol. Typically, the symbol size is proportional to the sample size used to calculate the estimator. If NULL, the same symbol size is used for all subgroups. By default tries to get this from tbl attribute col_symbol_size, otherwise needs to be manually specified.

col	(character) color(s).
ggtheme	(theme) a graphical theme as provided by ggplot2 to control styling of the plot.
as_list	(flag) whether the two ggplot objects should be returned as a list. If TRUE, a named list with two elements, table and plot, will be returned. If FALSE (default) the table and forest plot are printed side-by-side via cowplot::plot_grid().

Value

ggplot forest plot and table.

---

keep_by_label *Filter Teal Modules by Label*

Description

[Experimental] Recursively filters a teal modules object to keep only modules whose labels match the specified labels. Removes modules that don't match and empty parent containers.

Usage

```
keep_by_label(x, label)
```

Arguments

x	(teal_module or teal_modules) the object to filter.
label	(character(1)) character vector of module labels to keep.

Value

Filtered teal_modules or teal_module object, or NULL if none matches.

Examples

```
# Keep only specific modules by label
mods <- teal::modules(
  teal::example_module("mod1"),
  teal::example_module("mod2")
)
filtered_mods <- keep_by_label(mods, c("Data Table", "Disposition"))
```

`merge_levels_transformator` *Combine levels of a variable into one level*

Description

[Experimental]

Usage

```
merge_levels_transformator(dataname, predefined = list())
```

Arguments

`dataname`	(`character(1)`) the name of the dataset which columns will be used for possible transformation.
`predefined`	(`list`) the list which has variable name, levels and new label

Details

This transformator allows the user to select a column from the dataset and merge levels of this column into a new level. Only selected levels are affected.

Value

`teal::teal_transform_module`

Examples

```
app <- teal::init(
  data = teal.data::teal_data(IRIS = iris, code = "IRIS <- iris"),
  modules = teal::modules(
    teal::example_module(
      transformators = list(merge_levels_transformator(
        dataname = "IRIS",
        predefined = list(
          list("Species", "setosa", "SETOSA_WITHIN_FIX"),
          list("Petal.Width", c(0.2, 0.3, 0.5), 12)
        )
      ))
    )
  )
)
if (interactive()) {
  shiny::shinyApp(app$ui, app$server)
}
```

`or_filtering_transformator` *Apply Logical AND/OR filter transformations*

Description

[Experimental] This transformator provides users with flexible, dynamic filtering capabilities for datasets.

Each instance of the transformator operates on a **single dataset**. Users can define multiple filter blocks, where:

- Conditions within each block are combined with **logical AND**.
- Multiple blocks are combined with **logical OR**.

This allows creating complex filter expressions like:

`(Condition1 AND Condition2) OR (Condition3 AND Condition4)`

To apply filtering across **multiple datasets**, users can instantiate multiple instances of this module, each configured for a different dataset. Each module call is independent and manages filters for its specific dataset.

- **Supported Data Types & Expressions:**
  - Supports filtering on `character`, `factor`, and `numeric` columns.
  - Conditions can use operators: `==`, `!=`, `<`, `>`, `<=`, `>=`, `%in%`, `!%in%`.
  - Conditions are specified as simple expressions, e.g., `columnA == 'value' columnB != 5 columnC >= 10`
  - Each block's conditions are combined with **AND**.
  - Multiple blocks are combined with **OR**.
- **Features:**
  - **Add Multiple OR Blocks:** Dynamically add new blocks for alternative conditions.
  - **Add Conditions:** Within each block, add multiple conditions, with duplicate prevention.
  - **Preview Filter Expression:** Generate and display the current combined filter expression.
  - **Remove Conditions:** Remove individual conditions within a block.
  - **Expression Generation:** The resulting expression can be directly used with `dplyr::filter()` or similar functions.
- **Usage Pattern:**
  - Call the module multiple times with different dataset names to filter multiple datasets independently.
  - Each call manages its own filter state and expression.
  - Users can build complex filters per dataset and apply or combine them as needed.

Usage

```
or_filtering_transformator(dataname)
```

Arguments

dataname	(character(1)) Name of the dataset to filter. Pass a single dataset name as a string.

Value

teal::teal_transform_module

Examples

```
app <- teal::init(
  data = teal.data::teal_data(IRIS = iris),
  modules = teal::modules(teal::example_module(
    transformators = list(or_filtering_transformator("IRIS"))
  ))
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

patchwork_plot_decorator *Patchwork Decorator*

Description

[Experimental] Decorator function to add plot title and footnote to patchwork plots

Usage

```
patchwork_plot_decorator(output_name, label_text = "decorator")
```

Arguments

output_name	(character(1)) A name for the output plot object.
label_text	(character(1)) A customized label text for the decorator.

Details

The module creates a UI with text controls for plot title and footnote. The entered title and footnote text are applied to the patchwork plots.

Value

(teal.data::qenv) Returns a modified plot object with the transformation applied.

remove_by_label *Remove Teal Modules by Label*

Description

[Experimental] Recursively removes modules from a teal modules structure that match the specified labels.

Usage

```
remove_by_label(x, label)
```

Arguments

x	(teal_module or teal_modules) The object to filter.
label	(character(1)) character vector of module labels to remove.

Value

The filtered teal modules object with matching modules removed, or NULL if all modules are removed.

Examples

```
mods <- teal::modules(
  teal::example_module("mod1"),
  teal::example_module("mod2")
)
# Remove a single module
filtered_mods <- remove_by_label(mods, "Deaths")

# Remove multiple modules
filtered_mods <- remove_by_label(mods, c("Deaths", "Lab Summary Table"))
```

title_footer_decorator *Title and Footer Decorator*

Description

[Experimental] A function to create a UI component for selecting a title and footer for tables or plots. It reads title information from a specified Excel file and allows users to choose a title from the provided options. It also provides user with flexibility to customize the title and footer according to their specific needs.

Usage

```
title_footer_decorator(
  output_name,
  titles_file,
  choices = NULL,
  selected = NULL
)
```

Arguments

output_name	(character(1)) a name for the output object (e.g., a plot or table).
titles_file	(character(1)) the path to an Excel file containing title and footer information. The function expects the titles to be in the first sheet named Sheet1.
choices	(character) an array of titles and footers, which are available for selection. Default NULL, indicates all titles and footers are available.
selected	(character(1)) the selected title or footer. Default NULL, indicates no title or footer is selected.

Details

The module creates a UI with a dropdown for selecting a title. Once a title is selected, it updates the output by either adding titles to a table or modifying a plot's title and caption accordingly. Additionally, it includes a checkbox that user can check to enable customization, allowing them to enter their own values for the title and footer in the designated input fields.

Value

teal::teal_transform_module()

See Also

For the exact Excel workbook layout expected by this function, see the package vignette: vignette("title-footer-decora
package = "uteals")

Examples

```
library(openxlsx)
library(teal.modules.general)

example_excel <- data.frame(
  `TABLE ID` = c(
    "DO_NOT_DELETE",
    "TSFAE01A", "TSFAE01A", "TSFAE01A",
    "TSFAE01B", "TSFAE01B"
  ),
  IDENTIFIER = c(
    "DO_NOT_DELETE",
    "TITLE", "FOOTNOTE1", "FOOTNOTE2",
    "TITLE", "FOOTNOTE1"
```

```
  ),
  TEXT = c(
    "DO_NOT_DELETE",
    "Adverse Events Summary A", "Source: Clinical Study Report", "Confidential",
    "Adverse Events Summary B", "Draft Version"
  ),
  stringsAsFactors = FALSE,
  check.names = FALSE
)

temp_titles <- tempfile(fileext = ".xlsx")
write.xlsx(example_excel, temp_titles, sheetName = "Sheet1", asTable = TRUE)
plot_module <- tm_g_scatterplot(
  label = "Scatter Plot",
  x = data_extract_spec(
    dataname = "IRIS",
    select = select_spec(
      choices = variable_choices(
        "IRIS", c("Sepal.Length", "Sepal.Width")
      ), selected = "Sepal.Length"
    )
  ),
  y = data_extract_spec(
    dataname = "IRIS",
    select = select_spec(
      choices = variable_choices(
        "IRIS", c("Petal.Length", "Petal.Width")
      ), selected = "Petal.Length"
    )
  ),
  decorators = list(
    plot = title_footer_decorator(
      "plot", temp_titles,
      choices = c("TSFAE01A", "TSFAE01B"), selected = NULL
    )
  )
)

# Initialize the teal app
app <- init(
  data = teal_data(IRIS = iris),
  modules = list(plot_module)
)

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

`tm_report_manager` *Report Manager Module*

Description

Report Manager Module

Usage

```
tm_report_manager(reports_path = "reports", auto_save = TRUE)
```

Arguments

reports_path	character. Absolute path where reports should be stored.
auto_save	logical. Whether to save active report whenever there are any changes made.

Details

This module supports collaborative work on `teal` reports, generated by `teal.reporter` package. This module extends functionalities of `teal.reporter` and allows to store reports in chosen path. Reports will be stored as JSON files inside folders named after reports.

Examples

```
app <- teal::init(
  data = teal.data::teal_data(IRIS = iris),
  modules = teal::modules(
    teal::example_module(transformators = list(or_filtering_transformator("IRIS"))),
    tm_report_manager()
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}
```

---

use_freeze_tables *Freeze table headers and first columns*

Description

[Experimental] Makes the header row and/or the first column of tables rendered with `teal.widgets::table_with_setting` sticky, so that the headers and row labels stay visible while scrolling. Include the returned tags once in the app UI, e.g. in the `header` argument of `teal::init()`.

Usage

```
use_freeze_tables(
  freeze_header = TRUE,
  freeze_col = TRUE,
  max_height = "70vh",
  module_ids = NULL
)
```

Arguments

freeze_header	(logical(1)) whether to freeze the header rows of the tables.
freeze_col	(logical(1)) whether to freeze the first column of the tables.
max_height	(character(1)) a CSS value for the maximum height of the table container. The table scrolls vertically inside the container when it is taller than that.
module_ids	(character or NULL) if NULL, all tables in the app are frozen. Otherwise, only tables inside elements whose id contains any of the provided strings are frozen (e.g. the ids of teal modules).

Details

rtables renders tables without a <thead> element, so the header rows cannot be selected with CSS alone. The JavaScript watches the page for new tables and moves the leading rows containing <th> cells into a <thead> element of every matching table. Once https://github.com/pharmaverse/rtables/issues/1123 is resolved, this can be simplified.

Value

A shiny::tagList() with the CSS and the JavaScript freezing the tables.

Examples

```
library(teal.modules.clinical)
data <- teal_data()
data <- within(data, {
  ADSL <- tmc_ex_adsl
})
join_keys(data) <- default_cdisc_join_keys[names(data)]

app <- init(
  data = data,
  modules = modules(
    tm_t_summary(
      label = "Demographic Table",
      dataname = "ADSL",
      arm_var = choices_selected(c("ARM", "ARMCD"), "ARM"),
      add_total = TRUE,
      summarize_vars = choices_selected(
        c("SEX", "RACE", "BMRKR2", "EOSDY", "DCSREAS", "AGE"),
        c("SEX", "RACE")
      ),
      useNA = "ifany"
    )
  ),
  header = use_freeze_tables(freeze_header = TRUE, freeze_col = TRUE)
)

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

`watermark_decorator` *Watermark Decorator*

Description

[Experimental] A function to create a UI component for selecting watermark text for plots. Note: Currently tables are not supported

Usage

```
watermark_decorator(output_name, watermark_text = "", font_size = 90)
```

Arguments

`output_name`	(`character(1)`) a name for the output object (e.g., a plot or table).
`watermark_text`	(`character(1)`) text to display for the watermark.
`font_size`	(`character(1)`) font size for the watermark text.

Details

The module creates a UI with `textInput` for specifying watermark text and font size. the entered watermark text is displayed with a default `gridify` layout.

Value

`teal::teal_transform_module()`

Index

cowplot::plot_grid(), *11*
create_rel_risk_transformator, 3

extract_modules_to_yaml, 4

forestplot_x_decorator, 5

g_forest_with_transform, 9
ggplot2::ggplot(), *9*
ggplot_decorator, 7

keep_by_label, 11

merge_levels_transformator, 12

or_filtering_transformator, 13

patchwork_plot_decorator, 14

remove_by_label, 15
rtables::rtable(), *9*

shiny::tagList(), *19*

teal.modules.clinical::tm_g_forest_rsp, *5*
teal.widgets::table_with_settings_ui(), *18*
teal::init(), *18*
teal::teal_transform_module, *5*
teal::teal_transform_module(), *16*, *20*
title_footer_decorator, 15
tm_report_manager, 17

use_freeze_tables, 18
uteals *(*uteals-package*)*, 2
uteals-package, 2

watermark_decorator, 20