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

September 10, 2024

Type Package

Title RNA-Seq Analysis Modules to Add to a Teal Application

Version 0.1.6

Date 2024-02-13

Description RNA-seq analysis teal modules based on the `hermes` package.

License Apache License 2.0 | file LICENSE

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

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

Depends ggplot2, R (>= 4.1), shiny, teal (>= 0.15.0)

Imports checkmate, DT, edgeR, forcats, hermes (>= 1.7.1), lifecycle, logger (>= 0.2.0), MultiAssayExperiment, rtables (>= 0.5.1), S4Vectors, shinyRadioMatrix (>= 0.2.1), shinyWidgets, stats, stringr, SummarizedExperiment, teal.data (>= 0.3.0.9018), teal.logger (>= 0.1.1), teal.reporter (>= 0.2.0), teal.widgets (>= 0.4.0), tern (>= 0.7.10)

Suggests BiocStyle, covr, dplyr, globals, knitr, matrixStats, R6, rmarkdown, rvest, shinytest2 (>= 0.2.0), testthat (>= 2.0)

VignetteBuilder knitr

RdMacros lifecycle

Config/Needs/website insightsengineering/nesttemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

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

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

RemoteRef v0.1.6

RemoteSha 2970b6374e6282ec62bd240f3f01b322e50d7d00

Contents

`teal.modules.hermes-package` `teal.modules.hermes` *Package*

Description

`teal.modules.hermes` provides RNA-seq analysis modules to add to a teal application.

Author(s)

Maintainer: Daniel Sabanés Bové <daniel.sabanes_bove@roche.com>

Authors:

- Nikolas Burkoff
- Dinakar Kulkarni
- Konrad Pagacz
- Namrata Bhatia
- Dawid Kaledkowski
- Pawel Rucki
- Jeff Luong
- Stefanie Bienert
- Haocheng Li
- Lyndsee Midori Zhang

Other contributors:

- Sorin Voicu [contributor]
- Benoit Falquet [contributor]
- Mahmoud Hallal [contributor]
- Tim Treis [contributor]
- Vedha Viyash [contributor]
- F. Hoffmann-La Roche AG [copyright holder, funder]

See Also

Useful links:

- https://insightsengineering.github.io/teal.modules.hermes/
- https://github.com/insightsengineering/teal.modules.hermes/
- Report bugs at https://github.com/insightsengineering/teal.modules.hermes/issues

adtteSpecInput *Module Input for* ADTTE *Specification*

Description

[Experimental]

This defines the input for the ADTTE specification.

Usage

```
adtteSpecInput(inputId, label_paramcd = "Select Endpoint")
```

Arguments

inputId	(string) the ID used to call the module input.
label_paramcd	(string) label for the endpoint (PARAMCD) selection.

Value

The UI part.

See Also

adtteSpecServer() for the module server and a complete example.

adtteSpecServer *Module Server for* ADTTE *Specification*

Description

[Experimental]

This defines the server part for the ADTTE specification. The resulting data set binned_adtte_subset contains the subset of ADTTE selected by the time-to-event endpoint, joined together with the gene information extracted from specified assay and experiment, as numeric and factor columns. The factor column is created by binning the numeric column according to the quantile cutoffs specified in probs.

Usage

```
adtteSpecServer(
  id,
  data,
  mae_name,
  adtte_name,
  adtte_vars,
  experiment_data,
  experiment_name,
  assay,
  genes,
  probs
)
```

Arguments

id	(string) the shiny module id.
data	(reactive) reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.
mae_name	(string) name of the MAE data used in the teal module.
adtte_name	(string) name of the ADTTE dataset.
adtte_vars	(named list of string) names of the variables to use in the ADTTE dataset. It should comprise elements: • aval: the numeric time-to-event variable. • avalu: the variable holding the unit of aval. • is_event: the logical event variable. It needs to be TRUE when there was an observed event, and FALSE if the time is censored without observed event. • paramcd: the character or factor parameter code variable, defining the type of time-to-event for selection in the module. • usubjid: the subject ID variable.
experiment_data	(reactive AnyHermesData) input experiment.
experiment_name	(reactive string) name of the input experiment.
assay	(reactive string) name of the assay.
genes	(reactive GeneSpec) gene specification.
probs	(reactive numeric) probabilities to bin the gene or gene signature into.

Value

List with the following elements:

- `binned_adtte_subset`: reactive containing the joined `ADTTE` and gene data.
- `gene_col`: reactive containing the string with the column name of the original numeric gene variable.
- `gene_factor`: string with the variable name for the binned gene data.
- `time_unit`: reactive string with the time unit for the current subset.

See Also

`adtteSpecInput()` for the module UI.

Examples

```
ui <- function(id) {
  ns <- NS(id)

  teal.widgets::standard_layout(
    encoding = uiOutput(ns("encoding_ui")),
    output = verbatimTextOutput(ns("summary"))
  )
}

server <- function(id, data, filter_panel_api) {
  checkmate::assert_class(data, "reactive")
  checkmate::assert_class(shiny::isolate(data()), "teal_data")
  moduleServer(id, function(input, output, session) {
    output$encoding_ui <- renderUI({
      div(
        experimentSpecInput(session$ns("experiment"), data, mae_name = "MAE"),
        assaySpecInput(session$ns("assay")),
        geneSpecInput(session$ns("genes"), funs = list(Mean = colMeans)),
        adtteSpecInput(session$ns("adtte"))
      )
    })
    experiment <- experimentSpecServer(
      "experiment",
      data = data,
      filter_panel_api = filter_panel_api,
      mae_name = "MAE"
    )
    assay <- assaySpecServer(
      "assay",
      assays = experiment$assays
    )
    genes <- geneSpecServer(
      "genes",
      funs = list(Mean = colMeans),
      gene_choices = experiment$genes
    )
```

```
    adtte <- adtteSpecServer(
      "adtte",
      data = data,
      adtte_name = "ADTTE",
      mae_name = "MAE",
      adtte_vars = list(
        aval = "AVAL",
        avalu = "AVALU",
        is_event = "is_event",
        paramcd = "PARAMCD",
        usubjid = "USUBJID"
      ),
      experiment_data = experiment$data,
      experiment_name = experiment$name,
      assay = assay,
      genes = genes,
      probs = reactive({
        0.5
      })
    )
    output$summary <- renderPrint({
      binned_adtte_subset <- adtte$binned_adtte_subset()
      summary(binned_adtte_subset)
    })
  })
}

my_app <- function() {
  data <- teal_data()
  data <- within(data, {
    ADSL <- teal.data::rADSL
    ADTTE <- teal.modules.hermes::rADTTE %>%
      dplyr::mutate(is_event = .data$CNSR == 0)
    MAE <- hermes::multi_assay_experiment
  })
  datanames <- c("ADSL", "ADTTE", "MAE")
  datanames(data) <- datanames
  join_keys(data) <- default_cdisc_join_keys[datanames]

  app <- init(
    data = data,
    modules = modules(
      module(
        label = "adtteSpec example",
        server = server,
        ui = ui,
        datanames = "all"
      )
    )
  )
  shinyApp(app$ui, app$server)
}
```

```
if (interactive()) {
  my_app()
}
```

`assaySpecInput` *Module Input for Assay Specification*

Description

[Experimental]

This defines the input for the assay specification.

Usage

```
assaySpecInput(inputId, label_assays = "Select Assay")
```

Arguments

`inputId`	(`string`) the ID used to call the module input.
`label_assays`	(`string`) label for the assay selection.

Value

The UI part.

See Also

`assaySpecServer()` for the module server and a complete example.

`assaySpecServer` *Module Server for Assay Specification*

Description

[Experimental]

This defines the server part for the assay specification.

Usage

```
assaySpecServer(id, assays, exclude_assays = character())
```

Arguments

id	(string) the shiny module id.
assays	(reactive character) available assays in the currently selected experiment.
exclude_assays	(character) names of the assays which should not be included in choices in the teal module.

Value

The chosen assay as a reactive string.

See Also

assaySpecInput() for the module UI.

Examples

```
ui <- function(id) {
  ns <- NS(id)
  teal.widgets::standard_layout(
    encoding = uiOutput(ns("encoding_ui")),
    output = textOutput(ns("result"))
  )
}

server <- function(id, data, filter_panel_api) {
  moduleServer(id, module = function(input, output, session) {
    output$encoding_ui <- renderUI({
      div(
        experimentSpecInput(session$ns("experiment"), data, "MAE"),
        assaySpecInput(
          session$ns("assay"),
          label_assays = "Please choose assay"
        )
      )
    })
    experiment <- experimentSpecServer(
      id = "experiment",
      data = data,
      filter_panel_api = filter_panel_api,
      mae_name = "MAE"
    )
    assay <- assaySpecServer(
      "assay",
      experiment$assays,
      exclude_assays = c("counts", "cpm", "tpm", "bla")
    )
    output$result <- renderPrint({
      assay()
    })
  })
```

```
}

my_app <- function() {
  data <- teal_data(MAE = hermes::multi_assay_experiment)
  app <- init(
    data = data,
    modules = modules(
      module(
        label = "assaySpec example",
        server = server,
        ui = ui,
        datanames = "all"
      )
    )
  )
  shinyApp(app$ui, app$server)
}
if (interactive()) {
  my_app()
}
```

`assert_adtte_vars` *Check for* `ADTTE` *Variables*

Description

[Experimental]

Check whether `x` is a list of `ADTTE` variables.

Usage

```
assert_adtte_vars(x)
```

Arguments

`x`	an object to check.

See Also

`assertions` for more details.

Examples

```
assert_adtte_vars(list(aval = "AV", is_event = "EV", paramcd = "PC", usubjid = "ID", avalu = "u"))
```

assert_summary_funs *Check for List of Summary Functions*

Description

[Experimental]

Check whether x is a list of summary functions.

Usage

```
assert_summary_funs(x, null.ok = FALSE)
```

Arguments

x	an object to check.
null.ok	(flag) whether x may also contain NULL, meaning that a user choice is possible where no summary function should be applied.

See Also

assertions for more details.

Examples

```
assert_summary_funs(list(mean = colMeans, raw = NULL), null.ok = TRUE)
```

check_reactive *Check for Reactive Input*

Description

[Experimental]

Check whether x is a reactive input.

Usage

```
check_reactive(x)

assert_reactive(x, .var.name = checkmate::vname(x), add = NULL)

test_reactive(x)
```

Arguments

x	an object to check.
.var.name	(string) name of the checked object to print in assertions; defaults to the heuristic implemented in checkmate::vname().
add	(AssertCollection or NULL) collection to store assertion messages, see checkmate::AssertCollection.

See Also

assertions for more details.

Examples

```
check_reactive("bla")
check_reactive(reactive("bla"))
```

---

check_tag *Check for Shiny Tag*

Description

[Experimental]

Check whether x is a shiny tag.

Usage

```
check_tag(x, null.ok = FALSE)

assert_tag(x, null.ok = FALSE, .var.name = checkmate::vname(x), add = NULL)

test_tag(x, null.ok = FALSE)

expect_tag(x, null.ok = FALSE, info = NULL, label = vname(x))
```

Arguments

x	an object to check.
null.ok	(flag) whether x may also be NULL.
.var.name	(string) name of the checked object to print in assertions; defaults to the heuristic implemented in checkmate::vname().
add	(AssertCollection or NULL) collection to store assertion messages, see checkmate::AssertCollection.

info (string)
extra information to be included in the message for the testthat reporter, see testthat::expect_that().

label (string)
name of the checked object to print in messages. Defaults to the heuristic implemented in checkmate::vname().

See Also

assertions for more details.

Examples

```
check_tag("bla")
check_tag(NULL, null.ok = TRUE)
```

---

experimentSpecInput *Module Input for Experiment Specification*

Description

[Experimental]

This defines the input for the experiment specification.

Usage

```
experimentSpecInput(
  inputId,
  data,
  mae_name,
  label_experiments = "Select Experiment"
)
```

Arguments

inputId (string)
the ID used to call the module input.

data (reactive)
reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.

mae_name (string)
name of the MAE data used in the teal module.

label_experiments
(string)
label for the experiment selection.

Value

The UI part.

See Also

`experimentSpecServer()` for the module server and a complete example.

`experimentSpecServer` *Module Server for Experiment Specification*

Description

[Experimental]

This defines the server part for the experiment specification.

Usage

```
experimentSpecServer(
  id,
  data,
  filter_panel_api,
  mae_name,
  name_annotation = "symbol",
  sample_vars_as_factors = TRUE,
  with_mae_col_data = TRUE
)
```

Arguments

`id`	(`string`) the shiny module id.
`data`	(`reactive`) `reactive(<teal_data>)` holding all the data sets provided during app initialization after going through the filters.
`filter_panel_api`	(`FilterPanelAPI`) object describing the actual filter panel API.
`mae_name`	(`string`) name of the MAE data used in the teal module.
`name_annotation`	(`string` or `NULL`) which annotation column to use as name to return in the `genes` data. If `NULL`, then the `name` column will be set to empty strings.
`sample_vars_as_factors`	(`flag`) whether to convert the sample variables (columns in `colData()` of the experiment) from character to factor variables.

`with_mae_col_data`
: (`flag`)
whether to include the `colData()` of the MAE into the experiment `colData()`.

Value

List with the following reactive objects:

- `data`: the `hermes::AnyHermesData` experiment.
- `name`: the name of the experiment as selected by the user.
- `genes`: a `data.frame` with the genes in `data`, with columns `id` and `name`.
- `assays`: the names of the assays in `data`.

See Also

`experimentSpecInput()` for the module UI.

Examples

```
ui <- function(id,
               mae_name) {
  ns <- NS(id)
  teal.widgets::standard_layout(
    encoding = uiOutput(ns("encoding_ui")),
    output = div(
      verbatimTextOutput(ns("summary")),
      verbatimTextOutput(ns("head"))
    )
  )
}

server <- function(id,
                   data,
                   filter_panel_api,
                   mae_name) {
  moduleServer(id, function(input, output, session) {
    output$encoding_ui <- renderUI({
      div(
        experimentSpecInput(
          session$ns("my_experiment"),
          data,
          mae_name,
          label_experiments = "Please choose experiment"
        ),
        selectInput(
          session$ns("property"),
          "Please choose property",
          c("data", "name", "genes", "assays")
        )
      )
    })
    experiment <- experimentSpecServer(
```

```
      "my_experiment",
      data,
      filter_panel_api,
      mae_name
    )
    result <- reactive({
      req(input$property)
      switch(input$property,
        data = experiment$data(),
        name = experiment$name(),
        genes = experiment$genes(),
        assays = experiment$assays()
      )
    })
    output$summary <- renderPrint({
      result <- result()
      hermes::summary(result)
    })
    output$head <- renderPrint({
      result <- result()
      utils::head(result)
    })
  })
}

my_app <- function() {
  data <- teal_data(MAE = hermes::multi_assay_experiment)
  app <- init(
    data = data,
    modules = modules(
      module(
        label = "experimentSpec example",
        server = server,
        server_args = list(mae_name = "MAE"),
        ui = ui,
        ui_args = list(mae_name = "MAE"),
        datanames = "all"
      )
    )
  )
  shinyApp(app$ui, app$server)
}
if (interactive()) {
  my_app()
}
```

---

geneSpecInput *Module Input for Gene Signature Specification*

Description

[Experimental]

This defines the input for the gene signature specification.

Usage

```
geneSpecInput(
  inputId,
  funs,
  label_genes = "Select Gene(s)",
  label_funs = "Select Gene Summary",
  label_text_button = "Enter list of genes",
  label_lock_button =
    "Lock gene selection (so that it does not get updated when filtering)",
  label_select_all_button = paste0("Select All Genes (first ", max_options, ")"),
  label_select_none_button = "Select None",
  max_options = 200L,
  max_selected = max_options
)
```

Arguments

`inputId`	(`string`) the ID used to call the module input.
`funs`	(named `list`) names of this list will be used for the function selection drop down menu.
`label_genes`	(`string`) label for the gene selection.
`label_funs`	(`string`) label for the function selection.
`label_text_button`	(`string`) label for the text input button.
`label_lock_button`	(`string`) label for the lock button.
`label_select_all_button`	(`string`) label for the selecting all genes button.
`label_select_none_button`	(`string`) label for the selecting no genes button.
`max_options`	(`count`) maximum number of gene options rendering and selected via "Select All".
`max_selected`	(`count`) maximum number of genes which can be selected.

Value

The UI part.

See Also

`geneSpecServer()` for the module server and a complete example.

Examples

```
geneSpecInput("my_genes", list(mean = colMeans), label_funs = "Please select function")
```

---

geneSpecServer *Module Server for Gene Signature Specification*

Description

[Experimental]

This defines the server part for the gene signature specification.

Usage

```
geneSpecServer(
  id,
  funs,
  gene_choices,
  label_modal_title = "Enter list of genes",
 label_modal_footer = c("Please enter a comma-separated list of gene IDs and/or names.",
    "(Note that genes not included in current choices will be removed)")
)
```

Arguments

`id`	(`string`) the shiny module id.
`funs`	(static named `list`) names of this list will be used for the function selection drop down menu.
`gene_choices`	(reactive `data.frame`) returns the possible gene choices to populate in the UI, as a `data.frame` with columns `id` and `name`.
`label_modal_title`	(`string`) title for the dialog that asks for the text input.
`label_modal_footer`	(`character`) lines of text to use for the footer of the dialog.

Value

Reactive hermes::GeneSpec which can be used as input for the relevant hermes functions.

See Also

geneSpecInput() for the module UI.

Examples

```
ui <- function(id, funs) {
  ns <- NS(id)
  teal.widgets::standard_layout(
    encoding = div(
      geneSpecInput(
        ns("my_genes"),
        funs = funs,
        label_funs = "Please select function"
      )
    ),
    output = textOutput(ns("result"))
  )
}
server <- function(id,
                   data,
                   funs) {
  checkmate::assert_class(data, "reactive")
  checkmate::assert_class(shiny::isolate(data()), "teal_data")
  moduleServer(id, function(input, output, session) {
    gene_choices <- reactive({
      mae <- data()[["MAE"]]
      object <- mae[[1]]
      gene_ids <- rownames(object)
      gene_names <- SummarizedExperiment::rowData(object)$symbol
      gene_data <- data.frame(
        id = gene_ids,
        name = gene_names
      )
      gene_data[order(gene_data$name), ]
    })
    gene_spec <- geneSpecServer(
      "my_genes",
      funs = funs,
      gene_choices = gene_choices
    )
    output$result <- renderText({
      validate_gene_spec(
        gene_spec(),
        gene_choices()$id
      )
      gene_spec <- gene_spec()
      gene_spec$get_label()
    })
```

```
  })
}
funs <- list(mean = colMeans)
my_app <- function() {
  data <- teal_data(MAE = hermes::multi_assay_experiment)
  app <- init(
    data = data,
    modules = modules(
      module(
        label = "GeneSpec example",
        server = server,
        server_args = list(funs = funs),
        ui = ui,
        ui_args = list(funs = funs),
        datanames = "all"
      )
    )
  )
  shinyApp(app$ui, app$server)
}
if (interactive()) {
  my_app()
}
```

`heatmap_plot` *Correlation Heatmap Plot*

Description

[Experimental]

This function plots the correlation heatmap.

Usage

```
heatmap_plot(object, assay_name)
```

Arguments

`object`	(`AnyHermesData`) contains RNA-seq values for one experiment.
`assay_name`	(`string`) the assay to define the groups.

Value

Plot to be displayed in the teal app.

Examples

```
library(hermes)
object <- HermesData(summarized_experiment)
result <- heatmap_plot(object, assay_name = "counts")
```

---

h_assign_to_group_list

Helper Function For Group List Creation

Description

[Experimental]

This helper function takes an assignment list and converts it to a group list.

Usage

```
h_assign_to_group_list(x)
```

Arguments

x	(named `list` of `character`) input assignment list.

Value

A combination list.

Examples

```
assign_list <- list(
  "ASIAN" = "1",
  "BLACK OR AFRICAN AMERICAN" = "1",
  "MULTIPLE" = "2",
  "UNKNOWN" = "2",
  "WHITE" = "4"
)
objective_list <- list(
  "ASIAN/BLACK OR AFRICAN AMERICAN" = c("ASIAN", "BLACK OR AFRICAN AMERICAN"),
  "MULTIPLE/UNKNOWN" = c("MULTIPLE", "UNKNOWN"),
  "WHITE" = "WHITE"
)
result_list <- h_assign_to_group_list(assign_list)
stopifnot(identical(result_list, objective_list))
```

h_collapse_levels *Helper Function for Collapsing of Factor Levels*

Description

[Experimental]

Given a group list and a factor, this helper function collapses the levels in the factor accordingly and also ensures that the resulting levels are in the order given in the group list.

Usage

```
h_collapse_levels(x, group_list)
```

Arguments

x	(factor) original factor.
group_list	(named list of character) includes the collapsing specification.

Value

The transformed factor x with new levels.

Examples

```
set.seed(123)
x <- factor(sample(
  c("ASIAN", "BLACK OR AFRICAN AMERICAN", "MULTIPLE", "UNKNOWN", "WHITE"),
  size = 30L,
  replace = TRUE
))
group_list <- list(
  "ASIAN/BLACK OR AFRICAN AMERICAN" = c("ASIAN", "BLACK OR AFRICAN AMERICAN"),
  "MULTIPLE/UNKNOWN" = c("MULTIPLE", "UNKNOWN"),
  "WHITE" = "WHITE"
)
x_collapsed <- h_collapse_levels(x, group_list)
stopifnot(identical(levels(x_collapsed), names(group_list)))
```

h_extract_words *Helper Function to Extract Words*

Description

[Experimental]

This helper function extracts words from a string. Here words are defined as containing lower or upper case letters, colons and dots. All other characters are considered separators.

Usage

```
h_extract_words(x)
```

Arguments

x	(string) input.

Value

Character vector with the extracted words.

Examples

```
h_extract_words("a, b, , c, 234; 34562 - GeneID:bla")
h_extract_words("GeneID:1820, sdf.393; 32596")
```

h_gene_data *Helper Function to Format Gene Choices*

Description

[Experimental]

Given a `hermes::AnyHermesData` data object, as well as the annotation column name to use as gene name, this function formats the contained genes as a `data.frame` ready for consumption in `h_order_genes()` e.g.

Usage

```
h_gene_data(object, name_annotation)
```

Arguments

object	(AnyHermesData) contains RNA-seq values for one experiment.
name_annotation	(string or NULL) which annotation column to use as name to return in the genes data. If NULL, then the name column will be set to empty strings.

Details

Note that missing names or names that only contain whitespace are replaced by empty strings for consistency and better labeling in the UI downstream

Value

A data.frame with id and name columns containing all genes from object.

Examples

```
object <- hermes::hermes_data[1:10, ]
h_gene_data(object, "symbol")
```

---

h_km_mae_to_adtte *Data Preprocessing for* ADTTE *Module*

Description

[Experimental]

A function to help with merging of MAE to ADTTE.

Usage

```
h_km_mae_to_adtte(
  adtte,
  mae,
  genes,
  experiment_name = "hd1",
  assay_name = "counts",
  usubjid_var = "USUBJID"
)
```

Arguments

adtte	(data frame) an adtte dataset.
mae	(MultiAssayExperiment) contains AnyHermesData objects.
genes	(GeneSpec) specification for gene(s) (signature), e.g. using hermes::gene_spec().
experiment_name	(string) the desired HermesData to use.
assay_name	(string) the assay to define the groups.
usubjid_var	(string) variable name of the subject ID variable.

Value

A data frame containing all columns/rows from adtte that match by subject ID with the row names of the MAE and have the gene samples available in the given experiment. The attribute gene_cols contains the column names for the gene columns.

Note

The final gene column names can start with a different string than the original gene IDs (or labels), in particular white space and colons are removed.

Examples

```
mae <- hermes::multi_assay_experiment
adtte <- teal.modules.hermes::rADTTE %>%
  dplyr::mutate(CNSR = as.logical(CNSR))

new_adtte <- h_km_mae_to_adtte(
  adtte,
  mae,
  genes = hermes::gene_spec("GeneID:1820"),
  experiment_name = "hd2"
)
new_adtte2 <- h_km_mae_to_adtte(
  adtte,
  mae,
  genes = hermes::gene_spec(c("GeneID:1820", "GeneID:94115"), fun = colMeans),
  experiment_name = "hd2"
)
new_adtte3 <- h_km_mae_to_adtte(
  adtte,
  mae,
  genes = hermes::gene_spec(c(A = "GeneID:1820", B = "GeneID:94115")),
  experiment_name = "hd2"
)
```

h_order_genes *Helper Function to Order Gene Choices*

Description

[Experimental]

The possible gene choices are ordered as follows. First come all genes which have a non-empty name, ordered by their name alphabetically. Last come all genes with an empty name, ordered by their ID alphabetically.

Usage

```
h_order_genes(genes)
```

Arguments

genes	(data.frame) containing id and name columns of the gene choices. Note that no missing values are allowed.

Value

The ordered data.frame.

Examples

```
genes <- data.frame(
  id = c("7", "1", "2", "345346", "0"),
  name = c("e", "", "c", "", "a")
)
h_order_genes(genes)
```

h_parse_genes *Helper Function to Parse Genes*

Description

[Experimental]

This helper function takes a vector of words and tries to match them with the id and name columns of possible gene choices.

Usage

```
h_parse_genes(words, choices)
```

Arguments

words	(character) containing gene IDs or names.
choices	(data.frame) containing id and name columns of the new choices.

Value

The subset of choices which matches words in ID or name.

Examples

```
h_parse_genes(
  c("a", "2535"),
  data.frame(id = as.character(2533:2537), name = letters[1:5])
)
```

---

h_update_gene_selection

Helper Function to Update Gene Selection

Description

[Experimental]

This helper function takes the intersection of selected and choices for genes and updates the inputId accordingly. It then shows a notification if not all selected genes were available.

Usage

```
h_update_gene_selection(session, inputId, selected, choices)
```

Arguments

session	(ShinySession) the session object.
inputId	(string) the ID used to call the module input.
selected	(character) currently selected gene IDs.
choices	(data.frame) containing id and name columns of the new choices.

is_blank *Checking for Empty String*

Description

[Experimental]

This predicate function is helpful for functions where arguments could not yet be initialized from the teal module.

Usage

```
is_blank(x)
```

Arguments

x	object to check.

Value

Flag whether x is identical to an empty string, i.e. "".

Examples

```
is_blank("")
is_blank(" ")
```

multiSampleVarSpecServer *Module Server for Specification of Multiple Sample Variables*

Description

[Experimental]

When multiple sample variables are used in a given module, then this wrapper makes it much easier to specify in the server function.

Usage

```
multiSampleVarSpecServer(inputIds, original_data, ...)
```

Arguments

inputIds	(character) multiple input IDs corresponding to the different sample variables specified in the UI function.
original_data	(reactive SummarizedExperiment) input experiment where the sample variables extracted via SummarizedExperiment::colData() should be eligible for selection.
...	additional arguments as documented in sampleVarSpecServer(), namely the mandatory experiment_name and the optional categorical_only, num_levels and label_modal_title. transformed_data and assign_lists should not be specified as they are already specified internally here.

Value

List with the final transformed experiment_data reactive and a list vars which contains the selected sample variables as reactives under their input ID.

Examples

```
## Not run:
# In the server use:
sample_var_specs <- multiSampleVarSpecServer(
  inputIds = c("facet_var", "color_var"),
  experiment_name = reactive({
    input$experiment_name
  }),
  original_data = ori_data # nolint Please update the <ori_data>
)
# Then can extract the transformed data and selected variables later:
experiment_data <- sample_var_specs$experiment_data()
facet_var <- sample_var_specs$vars$facet_var()
color_var <- sample_var_specs$vars$color_var()

## End(Not run)
```

---

sampleVarSpecInput *Module Input for Sample Variable Specification*

Description

[Experimental]

This defines the input for the sample variable specification.

Usage

```
sampleVarSpecInput(
  inputId,
  label_vars = "Select sample variable",
  label_levels_button = "Combine factor levels"
)
```

Arguments

inputId	(string) the ID used to call the module input.
label_vars	(string) label for the sample variable selection.
label_levels_button	(string) label for the levels combination button.

Value

The UI part.

See Also

sampleVarSpecServer() for the module server and a complete example.

Examples

```
sampleVarSpecInput("my_vars", label_vars = "Select faceting variable")
```

---

sampleVarSpecServer *Module Server for Sample Variable Specification*

Description

[Experimental]

This defines the server part for the sample variable specification.

Usage

```
sampleVarSpecServer(
  id,
  experiment_name,
  original_data,
  transformed_data = original_data,
  assign_lists = reactiveValues(),
  num_levels = NULL,
```

```
  categorical_only = !is.null(num_levels),
  explicit_na = FALSE,
  label_modal_title = "Please click to group the original factor levels"
)
```

Arguments

id	(string) the shiny module id.
experiment_name	(reactive string) name of the input experiment.
original_data	(reactive SummarizedExperiment) input experiment where the sample variables extracted via SummarizedExperiment::colData() should be eligible for selection.
transformed_data	(reactive SummarizedExperiment) used when multiple sample variables can be selected in the app. In that case, pass here the pre-transformed data.
assign_lists	(reactivevalues) object to share factor level groupings across multiple sample variables.
num_levels	(count or NULL) required number of levels after combining original levels. If NULL then all numbers of levels are allowed.
categorical_only	(flag) whether only categorical variables should be selected from.
explicit_na	(flag) whether the colData of original_data will be transformed with hermes::h_df_factors_with_explicit_na before further processing. That means also that NA will be made an explicit factor level and counted for num_levels.
label_modal_title	(string) title for the dialog that asks for the text input.

Value

Reactive SummarizedExperiment::SummarizedExperiment which can be used as input for the relevant hermes functions.

Note

Only atomic columns (e.g. not DataFrame columns) of the colData which are not completely missing (NA) will be shown for selection. If num_levels is specified then only factor columns will be available.

See Also

sampleVarSpecInput() for the module UI.

Examples

```
ui <- function(id) {
  checkmate::assert_class(data, "teal_data")
  ns <- NS(id)

  teal.widgets::standard_layout(
    encoding = uiOutput(ns("encoding_ui")),
    output = plotOutput(ns("plot"))
  )
}
server <- function(id,
                   data) {
  checkmate::assert_class(data, "reactive")
  checkmate::assert_class(shiny::isolate(data()), "teal_data")
  moduleServer(id, function(input, output, session) {
    output$encoding_ui <- renderUI({
      mae <- data()[["MAE"]]
      experiment_name_choices <- names(mae)
      div(
       selectInput(session$ns("experiment_name"), "Select experiment", experiment_name_choices),
         sampleVarSpecInput(session$ns("facet_var"), "Select faceting variable")
      )
    })
    experiment_data <- reactive({
      req(input$experiment_name)
      mae <- data()[["MAE"]]
      object <- mae[[input$experiment_name]]
      SummarizedExperiment::colData(object) <-
        hermes::df_cols_to_factor(SummarizedExperiment::colData(object))
      object
    })
    facet_var_spec <- sampleVarSpecServer(
      "facet_var",
      experiment_name = reactive({
        input$experiment_name
      }),
      original_data = experiment_data
    )
    output$plot <- renderPlot({
      experiment_data_final <- facet_var_spec$experiment_data()
      facet_var <- facet_var_spec$sample_var()
      hermes::draw_boxplot(
        experiment_data_final,
        assay_name = "counts",
        genes = hermes::gene_spec(hermes::genes(experiment_data_final)[1]),
        facet_var = facet_var
      )
    })
  })
}
my_app <- function() {
  data <- teal_data(MAE = hermes::multi_assay_experiment)
```

```
  app <- init(
    data = data,
    modules = modules(
      module(
        label = "sampleVarSpec example",
        server = server,
        ui = ui,
        datanames = "all"
      )
    )
  )
  shinyApp(app$ui, app$server)
}
if (interactive()) {
  my_app()
}
```

---

tm_g_barplot *Teal Module for RNA-seq Barplot*

Description

[Experimental]

This module provides an interactive barplot for RNA-seq gene expression analysis.

Usage

```
tm_g_barplot(
  label,
  mae_name,
  exclude_assays = character(),
  summary_funs = list(Mean = colMeans, Median = matrixStats::colMedians, Max =
    matrixStats::colMaxs),
  pre_output = NULL,
  post_output = NULL
)

ui_g_barplot(id, mae_name, summary_funs, pre_output, post_output)

srv_g_barplot(
  id,
  data,
  filter_panel_api,
  reporter,
  mae_name,
  exclude_assays,
  summary_funs
)
```

```
sample_tm_g_barplot()
```

Arguments

label	(string) menu item label of the module in the teal app.
mae_name	(string) name of the MAE data used in the teal module.
exclude_assays	(character) names of the assays which should not be included in choices in the teal module.
summary_funs	(named list of functions or NULL) functions which can be used in the the gene signatures. For modules that support also multiple genes without summary, NULL can be included to not summarize the genes but provide all of them.
pre_output	(shiny.tag or NULL) placed before the output to put the output into context (for example a title).
post_output	(shiny.tag or NULL) placed after the output to put the output into context (for example the shiny::helpText() elements can be useful).
id	(string) the shiny module id.
data	(reactive) reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.
filter_panel_api	(FilterPanelAPI) object describing the actual filter panel API.
reporter	(Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_barplot(): sets up the user interface.
- srv_g_barplot(): sets up the server with reactive graph.
- sample_tm_g_barplot(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
  modules = modules(
    tm_g_barplot(
      label = "barplot",
```

```
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_barplot()
}
```

---

tm_g_boxplot *Teal Module for RNA-seq Boxplot*

Description

[Experimental]

This module provides an interactive boxplot for RNA-seq gene expression analysis.

Usage

```
tm_g_boxplot(
  label,
  mae_name,
  exclude_assays = character(),
 summary_funs = list(None = NULL, Mean = colMeans, Median = matrixStats::colMedians, Max
    = matrixStats::colMaxs),
  pre_output = NULL,
  post_output = NULL
)

ui_g_boxplot(id, mae_name, summary_funs, pre_output, post_output)

srv_g_boxplot(
  id,
  data,
  filter_panel_api,
  reporter,
  mae_name,
  exclude_assays,
  summary_funs
)

sample_tm_g_boxplot()
```

Arguments

label	(string) menu item label of the module in the teal app.
mae_name	(string) name of the MAE data used in the teal module.
exclude_assays	(character) names of the assays which should not be included in choices in the teal module.
summary_funs	(named list of functions or NULL) functions which can be used in the the gene signatures. For modules that support also multiple genes without summary, NULL can be included to not summarize the genes but provide all of them.
pre_output	(shiny.tag or NULL) placed before the output to put the output into context (for example a title).
post_output	(shiny.tag or NULL) placed after the output to put the output into context (for example the shiny::helpText() elements can be useful).
id	(string) the shiny module id.
data	(reactive) reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.
filter_panel_api	(FilterPanelAPI) object describing the actual filter panel API.
reporter	(Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_boxplot(): sets up the user interface.
- srv_g_boxplot(): sets up the server with reactive graph.
- sample_tm_g_boxplot(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
  modules = modules(
    tm_g_boxplot(
      label = "boxplot",
      mae_name = "MAE"
    )
  )
```

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

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_boxplot()
}
```

`tm_g_forest_tte` *Teal Module for Survival Forest Plot*

Description

[Experimental]

This module provides an interactive survival forest plot.

Usage

```
tm_g_forest_tte(
  label,
  adtte_name,
  mae_name,
  adtte_vars = list(aval = "AVAL", is_event = "is_event", paramcd = "PARAMCD", usubjid =
    "USUBJID", avalu = "AVALU"),
  exclude_assays = "counts",
  summary_funs = list(Mean = colMeans, Median = matrixStats::colMedians, Max =
    matrixStats::colMaxs),
  pre_output = NULL,
  post_output = NULL,
  plot_height = c(600L, 200L, 2000L),
  plot_width = c(1360L, 500L, 2000L)
)

ui_g_forest_tte(
  id,
  adtte_name,
  mae_name,
  summary_funs,
  pre_output,
  post_output
)

srv_g_forest_tte(
  id,
  data,
```

```
  filter_panel_api,
  reporter,
  adtte_name,
  mae_name,
  adtte_vars,
  exclude_assays,
  summary_funs,
  plot_height,
  plot_width
)

sample_tm_g_forest_tte()
```

Arguments

`label`	(`string`) menu item label of the module in the teal app.
`adtte_name`	(`string`) name of the `ADTTE` dataset.
`mae_name`	(`string`) name of the MAE data used in the teal module.
`adtte_vars`	(named `list` of `string`) names of the variables to use in the `ADTTE` dataset. It should comprise elements: • `aval`: the numeric time-to-event variable. • `avalu`: the variable holding the unit of `aval`. • `is_event`: the logical event variable. It needs to be `TRUE` when there was an observed event, and `FALSE` if the time is censored without observed event. • `paramcd`: the character or factor parameter code variable, defining the type of time-to-event for selection in the module. • `usubjid`: the subject ID variable.
`exclude_assays`	(`character`) names of the assays which should not be included in choices in the teal module.
`summary_funs`	(named `list` of functions or `NULL`) functions which can be used in the the gene signatures. For modules that support also multiple genes without summary, `NULL` can be included to not summarize the genes but provide all of them.
`pre_output`	(`shiny.tag` or `NULL`) placed before the output to put the output into context (for example a title).
`post_output`	(`shiny.tag` or `NULL`) placed after the output to put the output into context (for example the `shiny::helpText()` elements can be useful).
`plot_height`	(`list`) list of integers to set the default, minimum, and maximum plot height.
`plot_width`	(`list`) list of integers to set the default, minimum, and maximum plot width.

id	(string) the shiny module id.
data	(reactive) reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.
filter_panel_api	(FilterPanelAPI) object describing the actual filter panel API.
reporter	(Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_forest_tte(): sets up the user interface.
- srv_g_forest_tte(): sets up the server with reactive graph.
- sample_tm_g_forest_tte(): sample module function.

Examples

```
data <- teal_data()
data <- within(data, {
  ADTTE <- teal.modules.hermes::rADTTE %>%
    dplyr::mutate(is_event = .data$CNSR == 0)
  MAE <- hermes::multi_assay_experiment
})
datanames <- c("ADTTE", "MAE")
datanames(data) <- datanames
join_keys(data)["ADTTE", "ADTTE"] <- c("STUDYID", "USUBJID", "PARAMCD")

app <- init(
  data = data,
  modules = modules(
    tm_g_forest_tte(
      label = "forestplot",
      adtte_name = "ADTTE",
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_forest_tte()
}
```

tm_g_km *Teal Module for* Kaplan-Meier *Plot*

Description

[Experimental]

This teal module produces a grid style Kaplan-Meier plot for data with ADaM structure.

Usage

```
tm_g_km(
  label,
  adtte_name,
  mae_name,
  adtte_vars = list(aval = "AVAL", is_event = "is_event", paramcd = "PARAMCD", usubjid =
    "USUBJID", avalu = "AVALU"),
  exclude_assays = "counts",
  summary_funs = list(Mean = colMeans, Median = matrixStats::colMedians, Max =
    matrixStats::colMaxs),
  pre_output = NULL,
  post_output = NULL
)

ui_g_km(id, adtte_name, mae_name, summary_funs, pre_output, post_output)

srv_g_km(
  id,
  data,
  filter_panel_api,
  reporter,
  adtte_name,
  mae_name,
  adtte_vars,
  summary_funs,
  exclude_assays
)

sample_tm_g_km()
```

Arguments

label	(string) menu item label of the module in the teal app.
adtte_name	(string) name of the ADTTE dataset.

`mae_name` (`string`)
name of the MAE data used in the teal module.

`adtte_vars` (named `list` of `string`)
names of the variables to use in the `ADTTE` dataset. It should comprise elements:

- `aval`: the numeric time-to-event variable.
- `avalu`: the variable holding the unit of `aval`.
- `is_event`: the logical event variable. It needs to be `TRUE` when there was an observed event, and `FALSE` if the time is censored without observed event.
- `paramcd`: the character or factor parameter code variable, defining the type of time-to-event for selection in the module.
- `usubjid`: the subject ID variable.

`exclude_assays` (`character`)
names of the assays which should not be included in choices in the teal module.

`summary_funs` (named `list` of functions or `NULL`)
functions which can be used in the the gene signatures. For modules that support also multiple genes without summary, `NULL` can be included to not summarize the genes but provide all of them.

`pre_output` (`shiny.tag` or `NULL`)
placed before the output to put the output into context (for example a title).

`post_output` (`shiny.tag` or `NULL`)
placed after the output to put the output into context (for example the `shiny::helpText()` elements can be useful).

`id` (`string`) the shiny module id.

`data` (`reactive`)
`reactive(<teal_data>)` holding all the data sets provided during app initialization after going through the filters.

`filter_panel_api`
(`FilterPanelAPI`)
object describing the actual filter panel API.

`reporter` (`Reporter`) object

Value

Shiny module to be used in the teal app.

Functions

- `ui_g_km()`: sets up the user interface.
- `srv_g_km()`: sets up the user interface.
- `sample_tm_g_km()`: sample module function.

Examples

```
data <- teal_data()
data <- within(data, {
  ADTTE <- teal.modules.hermes::rADTTE %>%
    dplyr::mutate(is_event = .data$CNSR == 0)
  MAE <- hermes::multi_assay_experiment
})
datanames <- c("ADTTE", "MAE")
datanames(data) <- datanames
join_keys(data)["ADTTE", "ADTTE"] <- c("STUDYID", "USUBJID", "PARAMCD")


modules <- modules(
  tm_g_km(
    label = "kaplan-meier",
    adtte_name = "ADTTE",
    mae_name = "MAE"
  )
)

app <- init(
  data = data,
  modules = modules
)

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

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_km()
}
```

---

tm_g_pca *Teal Module for PCA Analysis*

Description

[Experimental]

This module provides an interactive principal components plot and an interactive heatmap with correlation of principal components with sample variables.

Usage

```
tm_g_pca(
  label,
  mae_name,
  exclude_assays = character(),
```

```
  pre_output = NULL,
  post_output = NULL
)

ui_g_pca(id, mae_name, pre_output, post_output)

srv_g_pca(id, data, filter_panel_api, reporter, mae_name, exclude_assays)

sample_tm_g_pca()
```

Arguments

label	(string) menu item label of the module in the teal app.
mae_name	(string) name of the MAE data used in the teal module.
exclude_assays	(character) names of the assays which should not be included in choices in the teal module.
pre_output	(shiny.tag or NULL) placed before the output to put the output into context (for example a title).
post_output	(shiny.tag or NULL) placed after the output to put the output into context (for example the shiny::helpText() elements can be useful).
id	(string) the shiny module id.
data	(reactive) reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.
filter_panel_api	(FilterPanelAPI) object describing the actual filter panel API.
reporter	(Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_pca(): sets up the user interface.
- srv_g_pca(): sets up the server with reactive graph.
- sample_tm_g_pca(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
```

```
  modules = modules(
    tm_g_pca(
      label = "PCA plot",
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_pca()
}
```

`tm_g_quality` *Teal Module for RNA-seq Quality Control*

Description

[Experimental]

This module adds quality flags, filters by genes and/or samples, normalizes `AnyHermesData` objects and provides interactive plots for RNA-seq gene expression quality control.

Usage

```
tm_g_quality(
  label,
  mae_name,
  exclude_assays = character(),
  pre_output = NULL,
  post_output = NULL
)

ui_g_quality(id, mae_name, pre_output, post_output)

srv_g_quality(id, data, filter_panel_api, reporter, mae_name, exclude_assays)

sample_tm_g_quality()
```

Arguments

`label`	(`string`) menu item label of the module in the teal app.
`mae_name`	(`string`) name of the MAE data used in the teal module.

exclude_assays (character)
names of the assays which should not be included in choices in the teal module.

pre_output (shiny.tag or NULL)
placed before the output to put the output into context (for example a title).

post_output (shiny.tag or NULL)
placed after the output to put the output into context (for example the shiny::helpText() elements can be useful).

id (string) the shiny module id.

data (reactive)
reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.

filter_panel_api
(FilterPanelAPI)
object describing the actual filter panel API.

reporter (Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_quality(): sets up the user interface.
- srv_g_quality(): sets up the server with reactive graphs.
- sample_tm_g_quality(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
  modules = modules(
    tm_g_quality(
      label = "Quality Control",
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_quality()
}
```

`tm_g_scatterplot` *Teal Module for RNA-seq Scatterplot*

Description

[Experimental]

This module provides an interactive scatterplot for RNA-seq gene expression analysis.

Usage

```
tm_g_scatterplot(
  label,
  mae_name,
  exclude_assays = "counts",
  summary_funs = list(Mean = colMeans, Median = matrixStats::colMedians, Max =
    matrixStats::colMaxs),
  pre_output = NULL,
  post_output = NULL
)

ui_g_scatterplot(id, mae_name, summary_funs, pre_output, post_output)

srv_g_scatterplot(
  id,
  data,
  filter_panel_api,
  reporter,
  mae_name,
  exclude_assays,
  summary_funs
)

sample_tm_g_scatterplot()
```

Arguments

`label`	(`string`) menu item label of the module in the teal app.
`mae_name`	(`string`) name of the MAE data used in the teal module.
`exclude_assays`	(`character`) names of the assays which should not be included in choices in the teal module.
`summary_funs`	(named `list` of functions or `NULL`) functions which can be used in the the gene signatures. For modules that support also multiple genes without summary, `NULL` can be included to not summarize the genes but provide all of them.

pre_output (shiny.tag or NULL)
placed before the output to put the output into context (for example a title).

post_output (shiny.tag or NULL)
placed after the output to put the output into context (for example the shiny::helpText() elements can be useful).

id (string) the shiny module id.

data (reactive)
reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.

filter_panel_api
(FilterPanelAPI)
object describing the actual filter panel API.

reporter (Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_scatterplot(): sets up the user interface.
- srv_g_scatterplot(): sets up the server with reactive graph.
- sample_tm_g_scatterplot(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
  modules = modules(
    tm_g_scatterplot(
      label = "scatterplot",
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_scatterplot()
}
```

`tm_g_volcanoplot` *Teal Module for RNA-seq Volcano Plot*

Description

[Experimental]

This module provides an interactive volcano plot for RNA-seq gene expression analysis.

Usage

```
tm_g_volcanoplot(
  label,
  mae_name,
  exclude_assays = character(),
  pre_output = NULL,
  post_output = NULL
)

ui_g_volcanoplot(id, mae_name, pre_output, post_output)

srv_g_volcanoplot(
  id,
  data,
  filter_panel_api,
  reporter,
  mae_name,
  exclude_assays
)

sample_tm_g_volcanoplot()
```

Arguments

`label`	(`string`) menu item label of the module in the teal app.
`mae_name`	(`string`) name of the MAE data used in the teal module.
`exclude_assays`	(`character`) names of the assays which should not be included in choices in the teal module.
`pre_output`	(`shiny.tag` or `NULL`) placed before the output to put the output into context (for example a title).
`post_output`	(`shiny.tag` or `NULL`) placed after the output to put the output into context (for example the `shiny::helpText()` elements can be useful).
`id`	(`string`) the shiny module id.

data (reactive)
reactive(<teal_data>) holding all the data sets provided during app initialization after going through the filters.

filter_panel_api (FilterPanelAPI)
object describing the actual filter panel API.

reporter (Reporter) object

Value

Shiny module to be used in the teal app.

Functions

- ui_g_volcanoplot(): sets up the user interface.
- srv_g_volcanoplot(): sets up the server with reactive graph.
- sample_tm_g_volcanoplot(): sample module function.

Examples

```
data <- teal_data(MAE = hermes::multi_assay_experiment)
app <- init(
  data = data,
  modules = modules(
    tm_g_volcanoplot(
      label = "volcanoplot",
      mae_name = "MAE"
    )
  )
)
if (interactive()) {
  shinyApp(app$ui, app$server)
}

# Alternatively you can run the sample module with this function call:
if (interactive()) {
  sample_tm_g_volcanoplot()
}
```

---

top_gene_plot *Most Expressed Genes Plot*

Description

[Experimental]

This function plots the most expressed genes.

Usage

```
top_gene_plot(object, assay_name)
```

Arguments

object	(AnyHermesData) contains RNA-seq values for one experiment.
assay_name	(string) the assay to define the groups.

Value

Plot to be displayed in the teal app.

Examples

```
library(hermes)
object <- HermesData(summarized_experiment)
result <- top_gene_plot(object, assay_name = "counts")
```

---

validate_gene_spec *Validation of Gene Specification*

Description

[Experimental]

This validation function checks that a given hermes::GeneSpec has at least one gene selected and that all genes are included in possible choices.

Usage

```
validate_gene_spec(gene_spec, gene_choices)
```

Arguments

gene_spec	(GeneSpec) gene specification.
gene_choices	(character) all possible gene choices.

`validate_n_levels` *Validation of Number of Levels*

Description

[Experimental]

This validation function checks that a given vector `x` is a factor with the specified number of levels.

Usage

```
validate_n_levels(x, name, n_levels)
```

Arguments

`x`	(`factor`) factor to validate.
`name`	(`string`) name of `x` in the app.
`n_levels`	(`count`) required number of factor levels in `x`.

Index