Package 'iSEEu'

June 1, 2023

Type Package

Title iSEE Universe

Version 1.12.0

Date 2023-03-10

Description iSEEu (the iSEE universe) contains diverse functionality to extend the usage of the iSEE package, including additional classes for the panels, or modes allowing easy configuration of iSEE applications.

License MIT + file LICENSE

Encoding UTF-8

Depends iSEE, iSEEhex

Imports methods, S4Vectors, IRanges, shiny, SummarizedExperiment, SingleCellExperiment, ggplot2 (>= 3.4.0), DT, stats, colourpicker, shinyAce

Suggests scRNAseq, scater, scran, airway, edgeR, AnnotationDbi, org.Hs.eg.db, GO.db, KEGGREST, knitr, igraph, rmarkdown, BiocStyle, htmltools, Rtsne, uwot, testthat (>= 2.1.0), covr

URL https://github.com/iSEE/iSEEu

BugReports https://github.com/iSEE/iSEEu/issues

biocViews ImmunoOncology, Visualization, GUI, DimensionReduction, FeatureExtraction, Clustering, Transcription, GeneExpression, Transcriptomics, SingleCell, CellBasedAssays

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/iSEEu

git_branch RELEASE_3_17

git_last_commit 6521d6c

git_last_commit_date 2023-04-25

Date/Publication 2023-06-01

Author Kevin Rue-Albrecht [aut, cre] (<https://orcid.org/0000-0003-3899-3872>),
Charlotte Soneson [aut] (<https://orcid.org/0000-0003-3833-2169>),
Federico Marini [aut] (<https://orcid.org/0000-0003-3252-7758>),
Aaron Lun [aut] (<https://orcid.org/0000-0002-3564-4813>),
Michael Stadler [ctb]

Maintainer Kevin Rue-Albrecht <kevinrue67@gmail.com>

R topics documented:

`AggregatedDotPlot` *The AggregatedDotPlot class*

Description

Implements an aggregated dot plot where each feature/group combination is represented by a dot. The color of the dot scales with the mean assay value across all samples for a given group, while the size of the dot scales with the proportion of non-zero values across samples in that group.

Slot overview

The following slots control the choice of features:

- `CustomRows`, a logical scalar indicating whether custom rows in `CustomRowsText` should be used. If `TRUE`, the feature identities are extracted from the `CustomRowsText` slot; otherwise they are defined from a transmitted row selection. Defaults to `TRUE`.
- `CustomRowsText`, a string containing the names of the features of interest, typically corresponding to the row names of the SummarizedExperiment. Names should be new-line separated within this string. Defaults to the name of the first row in the SummarizedExperiment.

The following slots control the specification of groups:

- `ColumnDataLabel`, a string specifying the name of the `colData` field to use to group cells. The chosen field should correspond to a categorical factor. Defaults to the first categorical field.
- `ColumnDataFacet`, a string specifying the name of the `colData` field to use for faceting. The chosen field should correspond to a categorical factor. Defaults to `"---"`, i.e., no faceting.

The following slots control the choice of assay values:

- `Assay`, a string specifying the name of the assay containing continuous values, to use for calculating the mean and the proportion of non-zero values. Defaults to the first valid assay name.

The following slots control the visualization parameters:

- `VisualBoxOpen`, a logical scalar indicating whether the visual parameter box should be open on initialization. Defaults to `FALSE`.
- `VisualChoices`, a character vector specifying the visualization options to show. Defaults to `"Color"` but can also include `"Transform"` and `"Legend"`.

The following slots control the transformation of the mean values:

- `MeanNonZeroes`, a logical scalar indicating whether the mean should only be computed over non-zero values. Defaults to `FALSE`.
- `Center`, a logical scalar indicating whether the means for each feature should be centered across all groups. Defaults to `FALSE`.
- `Scale`, a logical scalar indicating whether the standard deviation for each feature across all groups should be scaled to unity. Defaults to `FALSE`.

The following slots control the color:

- `UseCustomColormap`, a logical scalar indicating whether to use a custom color scale. Defaults to `FALSE`, in which case the application-wide color scale defined by `ExperimentColorMap` is used.
- `CustomColorLow`, a string specifying the low color (i.e., at an average of zero) for a custom scale. Defaults to `"grey"`.
- `CustomColorHigh`, a string specifying the high color for a custom scale. Defaults to `"red"`.

- `CenteredColormap`, a string specifying the divergent colormap to use when `Center` is `TRUE`. Defaults to `"blue < grey < orange"`; other choices are `"purple < black < yellow"`, `"blue < grey < red"` and `"green < grey < red"`.

In addition, this class inherits all slots from its parent Panel class.

Constructor

`AggregatedDotPlot(...)` creates an instance of a AggregatedDotPlot class, where any slot and its value can be passed to `...` as a named argument.

Supported methods

In the following code snippets, `x` is an instance of an AggregatedDotPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- `.cacheCommonInfo(x)` adds a `"AggregatedDotPlot"` entry containing `continuous.assay.names` and `discrete.colData.names`.
- `.refineParameters(x, se)` returns `x` after setting `"Assay"`, `"ColumnDataLabel"` and `"ColumnDataFacet"` to valid values. If continuous assays or discrete `colData` variables are not available, `NULL` is returned instead.

For defining the interface:

- `.defineInterface(x, se, select_info)` creates an interface to modify the various parameters in the slots, mostly by calling the parent method and adding another visualization parameter box.
- `.defineDataInterface(x, se, select_info)` creates an interface to modify the data-related parameters, i.e., those that affect the position of the points.
- `.defineOutput(x)` defines the output HTML element.
- `.panelColor(x)` will return the specified default color for this panel class.
- `.fullName(x)` will return `"Aggregated dot plot"`.
- `.hideInterface(x)` will return `TRUE` for UI elements related to multiple row selections.

For monitoring reactive expressions:

- `.createObservers(x, se, input, session, pObjects, rObjects)` will create all relevant observers for the UI elements.

For generating output:

- `.generateOutput(x, se, all_memory, all_contents)` will return the aggregated dot plot as a ggplot object, along with the commands used for its creation.
- `.renderOutput(x, se, output, pObjects, rObjects)` will render the aggregated dot plot onto the interface.
- `.exportOutput(x, se, all_memory, all_contents)` will save the aggregated dot plot to a PDF file named after `x`, returning the path to the new file.

For providing documentation:

- `.definePanelTour(x)` will return a data.frame to be used in **rintrojs** as a panel-specific tour.

Author(s)

Aaron Lun

See Also

Panel, for the immediate parent class.

ComplexHeatmapPlot, for another panel with multi-row visualization capability.

Examples

```
library(scRNAseq)

# Example data ----
sce <- ReprocessedAllenData(assays="tophat_counts")
class(sce)

library(scater)
sce <- logNormCounts(sce, exprs_values="tophat_counts")

# launch the app itself ----
if (interactive()) {
    iSEE(sce, initial=list(
        AggregatedDotPlot(ColumnDataLabel="Primary.Type")
    ))
}
```

---

createGeneSetCommands *Create gene set commands*

Description

Create the commands required to populate FeatureSetTables with commonly used gene sets.

Usage

```
createGeneSetCommands(
  collections = c("GO", "KEGG"),
  organism = "org.Hs.eg.db",
  identifier = "ENTREZID"
)
```

Arguments

collections	Character vectors specifying the gene set collections of interest.
organism	String containing the **org.*.eg.db** package to use to extract mappings of gene sets to gene IDs.
identifier	String specifying the identifier to use to extract IDs for the organism package.

Details

GO terms are extracted using the "GOALL" mode, which extracts both direct and indirect children of each term. A description for each GO term is extracted using the **GO.db** package.

Mappings of genes to KEGG pathway are extracted from the organism package using the "PATH" term. Unfortunately, this is not up to date due to the licensing around KEGG terms. Descriptions for each pathway are extracted from http://rest.kegg.jp/list/pathway.

The output of this function can be used as the commands argument of registerFeatureSetCommands. It is also used by default in the FeatureSetTable constructor when no collections are registered.

Value

A list of two character vectors describing how to create collections and retrieve gene sets. This follows the expectations for commands in registerFeatureSetCommands.

Author(s)

Aaron Lun

See Also

FeatureSetTable, where the commands are intended for use.

registerFeatureSetCommands, to use the commands globally.

Examples

```
out <- createGeneSetCommands()
cat(out$collections['GO'], "\n")
cat(out$sets['GO'], "\n")
```

---

defunct *Defunct functions*

Description

Pretty much as it says here. These functions are all defunct and should not be used.

Usage

```
.getAcceptablePValueFields(...)

.getAcceptableAveAbFields(...)

.getAcceptableLogFCFields(...)

.setAcceptablePValueFields(...)
```

`.setAcceptableAveAbFields(...)`

`.setAcceptableLogFCFields(...)`

Arguments

...	Ignored.

Value

All functions error out with a defunct message pointing towards its replacement (if available).

Author(s)

Aaron Lun

DynamicMarkerTable-class

Dynamic marker table

Description

A table that dynamically identifies marker genes for a selected subset of samples. Comparisons are made between the active selection in the transmitting panel and (i) all non-selected points, if no saved selections are available; or (ii) each subset of points in each saved selection.

Slot overview

The following slots control the test procedure:

- `LogFC`, a numeric scalar indicating the log-fold change threshold to test against. Defaults to zero.
- `TestMethod`, string indicating the test to use (based on the `findMarkers` function from **scran**). This can be `"t"` (default), `"wilcox"` or `"binom"`.
- `Assay`, string indicating the assay to use for testing. Defaults to the first named assay in the SummarizedExperiment.

The following slots control the rendered table:

- `ExtraFields`, a character vector containing names of `rowData` columns to be included in the table. Set to the output of `getTableExtraFields`. This cannot be changed once the application starts.

In addition, this class inherits all slots from its parent RowTable, Table and Panel classes.

Constructor

`DynamicMarkerTable(...)` creates an instance of a DynamicMarkerTable class, where any slot and its value can be passed to ... as a named argument.

Supported methods

In the following code snippets, `x` is an instance of a DynamicMarkerTable class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- `.cacheCommonInfo(x)` adds a `"DynamicMarkerTable"` entry containing `valid.assay.names` and `valid.rowdata.names`. This will also call the equivalent RowTable method.
- `.refineParameters(x, se)` returns `x` after setting `"Assay"` to the first valid value. This will also call the equivalent RowTable method for further refinements to `x`. If valid assay names are not available, `NULL` is returned instead. Any `"ExtraFields"` are intersected with the valid `rowData` names.

For defining the interface:

- `.defineDataInterface(x, se, select_info)` returns a list of interface elements for manipulating all slots described above.
- `.panelColor(x)` will return the specified default color for this panel class.
- `.fullName(x)` will return `"Dynamic marker table"`.
- `.hideInterface(x)` will return `TRUE` for UI elements related to multiple row selections, otherwise calling the method for RowTable.

For monitoring reactive expressions:

- `.createObservers(x, se, input, session, pObjects, rObjects)` sets up observers for all new slots described above, as well as in the parent classes via the RowTable method.

For creating the table:

- `.generateTable(x, envir)` will create a data.frame of newly computed statistics in `envir`. The method will return the commands required to do so.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.

Examples

```
library(scRNAseq)
library(scater)

sce <- ReprocessedAllenData(assays="tophat_counts")
sce <- logNormCounts(sce, exprs_values="tophat_counts")
sce <- runPCA(sce, ncomponents=4)
sce <- runTSNE(sce)

dst <- DynamicMarkerTable(PanelId=1L, PanelWidth=8L,
    ColumnSelectionSource="ReducedDimensionPlot1")

rdp <- ReducedDimensionPlot(PanelId=1L,
    ColorByFeatureSource="DynamicMarkerTable1")
```

```
if (interactive()) {
    iSEE(sce, initial=list(rdp, dst))
}
```

DynamicReducedDimensionPlot-class
Dynamic reduced dimension plot

Description

A dimensionality reduction plot that dynamically recomputes the coordinates for the samples, based on the selected subset of samples (and possibly features) in transmitting panels. All samples in active and saved multiple selections are used here.

Slot overview

The following slots control the thresholds used in the visualization:

- `Type`, a string specifying the type of dimensionality reduction method to use. This can be `"PCA"` (default), `"TSNE"` or `"UMAP"`, which uses the relevant functions from the **scater** package.
- `NGenes`, an integer scalar specifying the number of highly variable genes to use in the dimensionality reduction. Only used if an explicit selection of features is not made in the app. Defaults to 1000.
- `Assay`, string indicating the assay to use for the calculations. Defaults to the first named assay in the SummarizedExperiment.

In addition, this class inherits all slots from its parent ColumnDotPlot, DotPlot and Panel classes.

Constructor

`DynamicReducedDimensionPlot(...)` creates an instance of a DynamicReducedDimensionPlot class, where any slot and its value can be passed to `...` as a named argument.

Supported methods

In the following code snippets, `x` is an instance of a DynamicReducedDimensionPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- `.cacheCommonInfo(x)` adds a `"DynamicReducedDimensionPlot"` entry containing `valid.assay.names`. This will also call the equivalent ColumnDotPlot method.
- `.refineParameters(x, se)` returns `x` after setting `"Assay"` to the first valid value. This will also call the equivalent ColumnDotPlot method for further refinements to `x`. If valid assay names are not available, `NULL` is returned instead.

For defining the interface:

- `.defineDataInterface(x, se, select_info)` returns a list of interface elements for manipulating all slots described above.
- `.panelColor(x)` will return the specified default color for this panel class.
- `.fullName(x)` will return `"Dynamic reduced dimension plot"`.

For monitoring reactive expressions:

- `.createObservers(x, se, input, session, pObjects, rObjects)` sets up observers for all new slots described above, as well as in the parent classes via the ColumnDotPlot method.

For creating the plot:

- `.generateDotPlotData(x, envir)` will create a data.frame of newly computed coordinates in `envir`. The method will return the commands required to do so as well as a list of labels.

For handling multiple selections:

- `.multiSelectionInvalidated(x)` will always return `TRUE`, as any change in the upstream selection of points will alter the coordinates and invalidate any brush/lasso on `x`.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.

Author(s)

Aaron Lun

Examples

```
library(scRNAseq)
library(scater)

sce <- ReprocessedAllenData(assays="tophat_counts")
sce <- logNormCounts(sce, exprs_values="tophat_counts")
sce <- runPCA(sce, ncomponents=4)
sce <- runTSNE(sce)

drdp <- DynamicReducedDimensionPlot(PanelId=1L, Assay="logcounts",
    ColumnSelectionSource="ReducedDimensionPlot1")

if (interactive()) {
    iSEE(sce, initial=list(ReducedDimensionPlot(PanelId=1L), drdp))
}
```

FeatureSetTable-class *Feature set table*

Description

A table where each row is itself a feature set and can be clicked to transmit a multiple feature selection to another panel. This relies on feature set collections that have been registered in the input SummarizedExperiment, see `registerFeatureSetCollections` for more details. If no collections have been registered, we default to the GO and KEGG collections from `createGeneSetCommands`.

Slot overview

The following slots control the feature sets in use:

- `Collection`, string specifying the type of feature set collection to show. Defaults to the first registered collection in the SummarizedExperiment.

The following slots control the table selections:

- `Selected`, a string containing the name of the currently selected gene set. Defaults to `""`, i.e., no selection.
- `Search`, a string containing the regular expression for the global search. Defaults to `""`, i.e., no search.
- `SearchColumns`, a character vector where each entry contains the search string for each column. Defaults to an empty character vector, i.e., no search.

In addition, this class inherits all slots from its parent Panel class.

Constructor

`FeatureSetTable(...)` creates an instance of a FeatureSetTable class, where any slot and its value can be passed to `...` as a named argument.

Supported methods

In the following code snippets, `x` is an instance of a FeatureSetTable class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- `.cacheCommonInfo(x)` adds a `"FeatureSetTable"` entry containing `available.sets`, a named list of DataFrames containing information about the individual gene sets for each collection. This will also call the equivalent Panel method.
- `.refineParameters(x, se)` replaces `NA` values in `Collection` with the first valid collection. It also replaces `NA` values for `Selected` with the first valid set in the chosen collection. This will also call the equivalent Panel method.

For defining the interface:

- `.defineDataInterface(x, se, select_info)` returns a list of interface elements for manipulating all slots described above.
- `.panelColor(x)` will return the specified default color for this panel class.
- `.fullName(x)` will return `"Gene set table"`.
- `.hideInterface(x)` will return `TRUE` for UI elements related to multiple selections, otherwise calling the method for Panel.
- `.defineOutput(x)` will return a HTML element containing a `datatable` widget.

For monitoring reactive expressions:

- `.createObservers(x, se, input, session, pObjects, rObjects)` sets up observers for all new slots described above, as well as in the parent classes via the Panel method.

For creating the table:

- `.generateOutput(x, envir)` will create a data.frame of gene set descriptions in `envir`. It will also return the commands required to do so and the name of the variable corresponding to said data.frame.
- `.renderOutput(x, se, ..., output, pObjects, rObjects)` will add a `datatable` widget to the output, which is used to render the aforementioned data.frame.

For controlling the multiple selections:

- `.multiSelectionDimension(x)` returns `"row"`.
- `.multiSelectionCommands(x, index)` returns a string specifying the commands to be used to extract the identities of the genes in the currently selected set. `index` is ignored.
- `.multiSelectionActive(x)` returns the name of the currently selected gene set, unless no selection is made, in which case `NULL` is returned.
- `.multiSelectionClear(x)` returns `x` but with the `Selected` slot replaced by an empty string.
- `.multiSelectionAvailable(x, contents)` returns `contents$available`, which is set to the number of features in `se`.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.

Author(s)

Aaron Lun

Examples

```
library(scRNAseq)
sce <- LunSpikeInData(location=FALSE)

library(scater)
sce <- logNormCounts(sce)

library(scran)
rowData(sce) <- cbind(rowData(sce), modelGeneVarWithSpikes(sce, "ERCC"))
```

```
cmds <- createGeneSetCommands(collections="GO",
    organism="org.Mm.eg.db", identifier="ENSEMBL")
sce <- registerFeatureSetCommands(sce, cmds)

# Setting up the application.
gst <- FeatureSetTable(PanelId=1L)

rdp <- RowDataPlot(RowSelectionSource="FeatureSetTable1",
    ColorBy="Row selection",
    XAxis="Row data", XAxisRowData="mean", YAxis="total")

rdt <- RowDataTable(RowSelectionSource="FeatureSetTable1")

if (interactive()) {
    iSEE(sce, initial=list(gst, rdp, rdt))
}
```

`GeneSetTable-class` *Gene set table*

Description

A table where each row is a gene set and can be clicked to transmit a multiple feature selection to another panel. This has been deprecated in favor of the simpler FeatureSetTable.

Slot overview

The following slots control the type of gene sets to show:

- `Type`, string specifying the type of gene set collection to show. Defaults to `"GO"`.

The following slots control the table selections:

- `Selected`, a string containing the name of the currently selected gene set. Defaults to `""`, i.e., no selection.
- `Search`, a string containing the regular expression for the global search. Defaults to `""`, i.e., no search.
- `SearchColumns`, a character vector where each entry contains the search string for each column. Defaults to an empty character vector, i.e., no search.

In addition, this class inherits all slots from its parent Panel class.

Constructor

`GeneSetTable(...)` creates an instance of a GeneSetTable class, where any slot and its value can be passed to `...` as a named argument.

Supported methods

In the following code snippets, x is an instance of a GeneSetTable class. Refer to the documentation for each method for more details on the remaining arguments.

For defining the interface:

- .defineDataInterface(x, se, select_info) returns a list of interface elements for manipulating all slots described above.
- .panelColor(x) will return the specified default color for this panel class.
- .fullName(x) will return "Gene set table".
- .hideInterface(x) will return TRUE for UI elements related to multiple selections, otherwise calling the method for Panel.
- .defineOutput(x) will return a HTML element containing a datatable widget.

For monitoring reactive expressions:

- .createObservers(x, se, input, session, pObjects, rObjects) sets up observers for all new slots described above, as well as in the parent classes via the Panel method.

For creating the table:

- .generateOutput(x, envir) will create a data.frame of gene set descriptions in envir, based on the mode="show" output of .getGeneSetCommands. It will also return the commands required to do so and the name of the variable corresponding to said data.frame.
- .renderOutput(x, se, ..., output, pObjects, rObjects) will add a datatable widget to the output, which is used to render the aforementioned data.frame.

For controlling the multiple selections:

- .multiSelectionDimension(x) returns "row".
- .multiSelectionCommands(x, index) returns a string specifying the commands to be used to extract the identities of the genes in the currently selected set, based on the mode="extract" output of .getGeneSetCommands. index is ignored.
- .multiSelectionActive(x) returns the name of the currently selected gene set, unless no selection is made, in which case NULL is returned.
- .multiSelectionClear(x) returns x but with the Selected slot replaced by an empty string.
- .multiSelectionAvailable(x, contents) returns contents$available, which is set to the number of features in se.

Author(s)

Aaron Lun

Examples

```
library(scRNAseq)
sce <- LunSpikeInData(location=FALSE)

library(scater)
sce <- logNormCounts(sce)

library(scran)
rowData(sce) <- cbind(rowData(sce), modelGeneVarWithSpikes(sce, "ERCC"))

# This defaults to 'org.Hs.eg.db' with 'ENTREZID'.
.setOrganism("org.Mm.eg.db")
.setIdentifierType("ENSEMBL")
gst <- GeneSetTable(PanelId=1L)

rdp <- RowDataPlot(RowSelectionSource="GeneSetTable1",
    ColorBy="Row selection",
    XAxis="Row data", XAxisRowData="mean", YAxis="total")

rdt <- RowDataTable(RowSelectionSource="GeneSetTable1")

if (interactive()) {
    iSEE(sce, initial=list(gst, rdp, rdt))
}
```

---

getPValuePattern *Global DE prefixes*

Description

Get or set patterns for acceptable names of `rowData` columns related to a differential expression analysis. These functions are deprecated; use their counterparts in ?"`registerPValuePatterns`" instead.

Usage

```
getPValuePattern()

getLogFCPattern()

getAveAbPattern()

setPValuePattern(value)

setLogFCPattern(value)

setAveAbPattern(value)
```

Arguments

value	A character vector containing the acceptable prefixes for each statistic.

Details

These utilities allow users to easily get and set the patterns of acceptable fields in all VolcanoPlots, MAPlots and LogFCLogFCPlots at once. Any global settings only take effect (i) during setup of the iSEE application and (ii) if the first panel of each class does not have existing values in the "PValueFields", "LogFCFields" or "AveAbFields" slots (which take precedence if present).

Each of these global settings are treated as *patterns* for partial matching. For the "PValue" pattern, columns with the names "PValue.X" and "X.PValue" will be considered acceptable matches. All partial matching must be exact - regular expressions are not supported.

Value

getPValuePattern returns the patterns for acceptable column names for p-values.

getLogFCPattern returns the patterns for acceptable column names for log-fold changes.

getAveAbPattern returns the patterns for acceptable column names for the average abundances.

The corresponding setters set the global parts for each statistic and return NULL invisibly.

Author(s)

Aaron Lun

See Also

VolcanoPlot, MAPlot and LogFCLogFCPlot, which are affected by these globals.

Examples

```
old <- getPValuePattern()

setPValuePattern(LETTERS)
getPValuePattern()

setPValuePattern(old)
```

---

getTableExtraFields *Global extra table fields*

Description

Get or set the names of the extra fields to include in a table.

Usage

```
getTableExtraFields()

setTableExtraFields(value)
```

Arguments

value	A character vector containing the names of extra fields to include.

Details

These utilities allow users to easily set the feature set commands for all DynamicMarkerTables at once. Any global settings only take effect (i) during setup of the iSEE application and (ii) if the first DynamicMarkerTable does not have an existing values in the "TableExtraFields" slots.

Value

getTableExtraFields returns the current global extra table fields.

setTableExtraFields will set the current global extra table fields and return NULL invisibly.

Author(s)

Aaron Lun

Examples

```
old <- getTableExtraFields()

setTableExtraFields(LETTERS)
getTableExtraFields()

setTableExtraFields(old)
```

---

iSEEu-pkg *iSEEu: iSEE Universe*

Description

iSEEu is a package that provides modes and panels for iSEE, allowing easy configuration of iSEE applications.

Author(s)

Charlotte Soneson <charlottesoneson@gmail.com>

Federico Marini <marinif@uni-mainz.de>

Kevin Rue-Albrecht <kevin.rue-albrecht@kennedy.ox.ac.uk>

Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

See Also

Useful links:

- https://github.com/iSEE/iSEEu
- Report bugs at https://github.com/iSEE/iSEEu/issues

`LogFCLogFCPlot-class` *The LogFCLogFCPlot class*

Description

The LogFCLogFCPlot is a RowDataPlot subclass that is dedicated to creating a scatter plot of two log-fold changes. Each axis contains the log-fold change for a differential expression analysis and each point represents a feature.

Slot overview

The following slots control the thresholds used in the visualization:

- `XPValueField`, a string specifying the field of `rowData` containing the p-values for the x-axis comparison.
- `YPValueField`, a string specifying the field of `rowData` containing the p-values for the y-axis comparison.
- `PValueThreshold`, a numeric scalar in (0, 1] specifying the threshold to use on the (adjusted) p-value. Defaults to 0.05.
- `LogFCThreshold`, a non-negative numeric scalar specifying the threshold to use on the log-fold change. Defaults to 0.
- `PValueCorrection`, a string specifying the multiple testing correction to apply. Defaults to `"BH"`, but can take any value from `p.adjust.methods`.

In addition, this class inherits all slots from its parent RowDataPlot, RowDotPlot, DotPlot and Panel classes.

Constructor

`LogFCLogFCPlot(...)` creates an instance of a LogFCLogFCPlot class, where any slot and its value can be passed to `...` as a named argument.

Users are expected to load relevant statistics into the `rowData` of a SummarizedExperiment. There should be two columns for the p-values from each comparison - and another two for the corresponding log-fold changes - for each gene/row, see Examples. The expected column names (and how to tune them) are listed at `?"registerPValueFields"`.

Supported methods

In the following code snippets, x is an instance of a RowDataPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- .cacheCommonInfo(x, se) returns se after being loaded with class-specific constants. This includes "valid.p.fields" and "valid.lfc.fields", character vectors containing the names of valid rowData columns for the p-values and log-fold changes, respectively.
- .refineParameters(x, se) returns x after setting XAxis="Row data" as well as "PValuePattern" and "LogFCPattern" to their corresponding cached values. This will also call the equivalent RowDataPlot method for further refinements to x. If valid p-value and log-fold change fields are not available, NULL is returned instead.

For defining the interface:

- .defineDataInterface(x, se, select_info) returns a list of interface elements for manipulating all slots described above.
- .panelColor(x) will return the specified default color for this panel class.
- .allowableXAxisChoices(x, se) returns a character vector specifying the acceptable log-fold change-related variables in rowData(se) that can be used as choices for the x-axis.
- .allowableYAxisChoices(x, se) returns a character vector specifying the acceptable log-fold change-related variables in rowData(se) that can be used as choices for the y-axis.
- .hideInterface(x, field) will return TRUE for field="XAxis", otherwise it will call the RowDataPlot method.
- .fullName(x) will return "LogFC-logFC plot".

For monitoring reactive expressions:

- .createObservers(x, se, input, session, pObjects, rObjects) sets up observers for all new slots described above, as well as in the parent classes via the RowDataPlot method.

For creating the plot:

- .generateDotPlotData(x, envir) will create a data.frame of row metadata variables in envir. This contains the two sets of log-fold changes on both axes, plus an extra field specifying whether or not the feature was considered to be significantly up or down. The method will return the commands required to do so as well as a list of labels.
- .prioritizeDotPlotData(x, envir) will create variables in envir marking the priority of points. Significant features receive higher priority (i.e., are plotted over their non-significant counterparts) and are less aggressively downsampled when Downsample=TRUE. The method will return the commands required to do this as well as a logical scalar indicating that rescaling of downsampling resolution is performed.
- .colorByNoneDotPlotField(x) will return a string specifying the field of the data.frame (generated by .generateDotPlotData) containing the significance information. This is to be used for coloring when ColorBy="None".
- .colorByNoneDotPlotScale(x) will return a string containing a **ggplot2** command to add a default color scale when ColorBy="None".

- `.generateDotPlot(x, labels, envir)` returns a list containing `plot` and `commands`, using the inital ColumnDataPlot ggplot and adding horizontal lines demarcating the log-fold change threshold.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.
- `.getDotPlotColorHelp(x, color_choices)` returns a function that generates an **rintrojs** tour for the color choice UI.

Author(s)

Aaron Lun

See Also

RowDataPlot, for the base class.

Examples

```
# Making up some results:
se <- SummarizedExperiment(matrix(rnorm(10000), 1000, 10))
rownames(se) <- paste0("GENE_", seq_len(nrow(se)))
rowData(se)$PValue1 <- runif(nrow(se))
rowData(se)$LogFC1 <- rnorm(nrow(se))
rowData(se)$PValue2 <- runif(nrow(se))
rowData(se)$LogFC2 <- rnorm(nrow(se))

if (interactive()) {
    iSEE(se, initial=list(LogFCLogFCPlot(XAxisRowData="LogFC1", YAxis="LogFC2",
        XPValueField="PValue1", YPValueField="PValue2")))
}
```

MAPlot-class *The MAPlot class*

Description

The MAPlot is a RowDataPlot subclass that is dedicated to creating a MA plot. It retrieves the log-fold change and average abundance and creates a row-based plot where each point represents a feature.

Slot overview

The following slots control the thresholds used in the visualization:

- `PValueField`, a string specifying the field of `rowData` containing the p-values.
- `PValueThreshold`, a numeric scalar in (0, 1] specifying the threshold to use on the (adjusted) p-value. Defaults to 0.05.

- LogFCThreshold, a non-negative numeric scalar specifying the threshold to use on the log-fold change. Defaults to 0.
- PValueCorrection, a string specifying the multiple testing correction to apply. Defaults to "BH", but can take any value from p.adjust.methods.

In addition, this class inherits all slots from its parent RowDataPlot, RowDotPlot, DotPlot and Panel classes.

Constructor

MAPlot(...) creates an instance of a MAPlot class, where any slot and its value can be passed to ... as a named argument.

Users are expected to load relevant statistics into the rowData of a SummarizedExperiment. This panel expects one or more columns containing the p-values, log-fold changes and average abundances for each gene/row - see Examples. The expected column names (and how to tune them) are listed at ?"registerPValueFields".

Supported methods

In the following code snippets, x is an instance of a RowDataPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- .cacheCommonInfo(x, se) returns se after being loaded with class-specific constants. This includes "valid.p.fields", "valid.ab.fields" and "valid.lfc.fields", which are character vectors containing the names of valid rowData columns for the p-values, average abundances and log-fold changes, respectively.
- .refineParameters(x, se) returns x after setting XAxis="Row data" and the various *Pattern fields to their cached values. This will also call the equivalent RowDataPlot method for further refinements to x. If valid p-value, abundance and log-fold change fields are not available, NULL is returned instead.

For defining the interface:

- .defineDataInterface(x, se, select_info) returns a list of interface elements for manipulating all slots described above.
- .panelColor(x) will return the specified default color for this panel class.
- .allowableXAxisChoices(x, se) returns a character vector specifying the acceptable average abundance-related variables in rowData(se) that can be used as choices for the x-axis.
- .allowableYAxisChoices(x, se) returns a character vector specifying the acceptable log-fold change-related variables in rowData(se) that can be used as choices for the y-axis.
- .hideInterface(x, field) will return TRUE for field="XAxis", otherwise it will call the RowDataPlot method.
- .fullName(x) will return "MA plot".

For monitoring reactive expressions:

- .createObservers(x, se, input, session, pObjects, rObjects) sets up observers for all new slots described above, as well as in the parent classes via the RowDataPlot method.

For creating the plot:

- `.generateDotPlotData(x, envir)` will create a data.frame of row metadata variables in `envir`. This should contain average abundances on the x-axis and log-fold changes on the y-axis, in addition to an extra field specifying whether or not the feature was considered to be significantly up or down. The method will return the commands required to do so as well as a list of labels.
- `.prioritizeDotPlotData(x, envir)` will create variables in `envir` marking the priority of points. Significant features receive higher priority (i.e., are plotted over their non-significant counterparts) and are less aggressively downsampled when `Downsample=TRUE`. The method will return the commands required to do this as well as a logical scalar indicating that rescaling of downsampling resolution is performed.
- `.colorByNoneDotPlotField(x)` will return a string specifying the field of the data.frame (generated by `.generateDotPlotData`) containing the significance information. This is to be used for coloring when `ColorBy="None"`.
- `.colorByNoneDotPlotScale(x)` will return a string containing a **ggplot2** command to add a default color scale when `ColorBy="None"`.
- `.generateDotPlot(x, labels, envir)` returns a list containing `plot` and `commands`, using the inital ColumnDataPlot ggplot and adding horizontal lines demarcating the log-fold change threshold.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.
- `.getDotPlotColorHelp(x, color_choices)` returns a function that generates an **rintrojs** tour for the color choice UI.

Author(s)

Aaron Lun

See Also

RowDataPlot, for the base class.

Examples

```
# Making up some results:
se <- SummarizedExperiment(matrix(rnorm(10000), 1000, 10))
rownames(se) <- paste0("GENE_", seq_len(nrow(se)))
rowData(se)$PValue <- runif(nrow(se))
rowData(se)$LogFC <- rnorm(nrow(se))
rowData(se)$AveExpr <- rnorm(nrow(se))

if (interactive()) {
    iSEE(se, initial=list(MAPlot()))
}
```

`MarkdownBoard-class` *The MarkdownBoard class*

Description

The MarkdownBoard class renders user-supplied Markdown into HTML to display inside the app. This is useful for displaying information alongside other panels, or for users to jot down their own notes.

Slot overview

The following slots are relevant to the rendered content:

- `Content`, a string containing Markdown-formatted text. This will be rendered to HTML for display inside the app.

In addition, this class inherits all slots from its parent Panel class.

Constructor

`MarkdownBoard(...)` creates an instance of a MarkdownBoard class, where any slot and its value can be passed to `...` as a named argument.

Supported methods

In the following code snippets, `x` is an instance of a RowDataPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For defining the interface:

- `.defineDataInterface(x, se, select_info)` returns a list of interface elements for editing the `Content`.
- `.panelColor(x)` will return the specified default color for this panel class.
- `.hideInterface(x, field)` will return `TRUE` for all selection-related parameters.
- `.fullName(x)` will return `"Volcano plot"`.

For monitoring reactive expressions:

- `.createObservers(x, se, input, session, pObjects, rObjects)` sets up observers for all new slots described above, as well as in the parent classes via the RowDataPlot method.

For rendering the display:

- `.defineOutput(x)` will return a UI element to display the HTML.
- `.renderOutput(x, se, ..., output, pObjects, rObjects)` will add reactive expressions to render the HTML.
- `.generateOutput(x, se, all_memory, all_contents)` will render the Markdown to HTML via the **rmarkdown** package, returning a string containing the rendered content in the `text` element of the output list. The Markdown-formatted content is converted into an R comment for code tracking purposes.

- `.exportOutput(x, se, all_memory, all_contents)` will create a HTML containing the rendered Markdown, and return a string containing the path to that HTML.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour. Not that there's a great deal to say here.

Author(s)

Aaron Lun

See Also

Panel, for the base class.

Examples

```
if (interactive()) {
    iSEE(SummarizedExperiment(), initial=list(MarkdownBoard()))
}
```

`modeEmpty` *App pre-configured to launch with no visible panel*

Description

This mode launches an app that does not display any panel.

Usage

```
modeEmpty(...)
```

Arguments

`...`	Arguments passed to `iSEE()`.

Details

This mode presents the advantage to launch an interface in a minimal amount of time, as it does not need to render any panel when the interface is launched. Users can then use the `"Organize panels"` widget to select panels to display in the interface.

Value

A Shiny app object is returned.

Examples

```
example("SingleCellExperiment")
rownames(sce) <- paste0("G", 1:200)
colnames(sce) <- paste0("C", 1:100)

app <- modeEmpty(sce)
if (interactive()) {
  shiny::runApp(app, port=1234)
}
```

`modeGating` *App pre-configured to link multiple feature assay plots*

Description

This mode launches a Shiny App preconfigured with multiple chain-linked feature expression plots for interactive data exploration of the SingleCellExperiment or `SummarizedExperiment` object.

Usage

```
modeGating(se, features, plotAssay = NA_character_, ..., plotWidth = 4)
```

Arguments

`se`	An object that coercible to SingleCellExperiment-class
`features`	`data.frame` with columns named `x` and `y` that define the features on the axes of the linked plots. Plots are serially linked from the first row to the last.
`plotAssay`	The assay (one of assayNames(se)) to use for the plots (character vector of length either 1 or equal to `nrow(features)`).
`...`	Additional arguments passed to `iSEE()`.
`plotWidth`	The grid width of linked plots (numeric vector of length either 1 or equal to `nrow(features)`

Value

A Shiny app object is returned.

Examples

```
library(scRNAseq)

# Example data ----
sce <- ReprocessedAllenData(assays="tophat_counts")
class(sce)

library(scater)
```

```
sce <- logNormCounts(sce, exprs_values="tophat_counts")

# Select top variable genes ----

plot_count <- 6
rv <- rowVars(assay(sce, "tophat_counts"))
top_var <- head(order(rv, decreasing=TRUE), plot_count*2)
top_var_genes <- rownames(sce)[top_var]

plot_features <- data.frame(
    x=head(top_var_genes, plot_count),
    y=tail(top_var_genes, plot_count),
    stringsAsFactors=FALSE
 )

# launch the app itself ----

app <- modeGating(sce, features = plot_features)
if (interactive()) {
  shiny::runApp(app, port=1234)
}
```

`modeReducedDim` *App pre-configured to compare multiple reduced dimension plots*

Description

This mode launches a Shiny App preconfigured with multiple linked reduced dimension plots for interactive data exploration of the `SingleCellExperiment` object.

Usage

```
modeReducedDim(
  se,
  includeNames = reducedDimNames(se),
  colorBy = NULL,
  ...,
  plotWidth = NULL
)
```

Arguments

`se`	An object that coercible to SingleCellExperiment
`includeNames`	Character vector with the names of reduced dimensions to display as individual panels. The default uses all available in `reducedDimNames(se)`.

colorBy	Character scalar controlling coloring of cells. Must match either to one of colnames(colData(se)) or rownames(se). If coloring by a colData column, a column data plot is opened in addition to the reduced dimension panels. If coloring by a feature, a row statistics table is openend in addition to the reduced dimension panels, from which the latter are receiving the color.
...	Additional arguments passed to iSEE.
plotWidth	The grid width of linked plots (numeric vector of length either 1 or equal to length(includeNames)). The total width of the window is 12, so plotWidth = 4 for example will show three panels per row. If plotWidth = NULL (the default), a value will be estimated depending on the number of reduced dimension panels.

Value

A Shiny app object is returned.

Examples

```
library(scRNAseq)

# Example data ----
sce <- ReprocessedAllenData(assays="tophat_counts")
class(sce)

library(scater)
sce <- logNormCounts(sce, exprs_values="tophat_counts")
sce <- runPCA(sce, ncomponents = 30)
sce <- runTSNE(sce)
sce <- runUMAP(sce)
reducedDimNames(sce)

# launch the app ----
# ... coloring by a column data variable
app <- modeReducedDim(sce, colorBy = "Primary.Type")
if (interactive()) {
    shiny::runApp(app, port=1234)
}
# ... coloring by a feature
app <- modeReducedDim(sce, colorBy = "Scnn1a")
if (interactive()) {
    shiny::runApp(app, port=1234)
}
```

---

registerDEFields *Register DE-related fields*

Description

Register the names of fields containing various DE statistics, to populate the user interface of DE-related Panels.

Usage

```
registerPValueFields(se, fields)

registerAveAbFields(se, fields)

registerLogFCFields(se, fields)

registerPValuePatterns(se, patterns)

registerAveAbPatterns(se, patterns)

registerLogFCPatterns(se, patterns)

getPValueFields(se)

getAveAbFields(se)

getLogFCFields(se)

getPValuePatterns(se, defaults = c("PValue", "p.value", "pval"))

getAveAbPatterns(se, defaults = c("AveExpr", "logCPM"))

getLogFCPatterns(se, defaults = c("logFC", "LogFC"))
```

Arguments

se	A SummarizedExperiment to be visualized with various DE-related Panels. This is expected to have a number of DE-related fields in its rowData.
fields	A character vector containing the names of the relevant fields containing the DE statistics. Alternatively NULL to remove any existing setting.
patterns	A character vector containing partial names, to match against the colnames of the rowData to identify relevant fields containing DE statistics. Alternatively NULL to remove any existing setting.
defaults	Character vector specifying the default patterns to provide when no patterns were registered in se.

Details

DE-related Panels need to find relevant rowData fields containing p-values, log-fold changes, etc. to set appropriate defaults in the user interface. These functions allow a user to tune the definition of what those Panels consider to be relevant, which is occasionally necessary if the DE statistics are stored in a rowData field with an unusual column name. The idea is to register the relevant fields in se, which can then be supplied to iSEE with the affected Panels - see Examples.

The registered fields should be the names of appropriate columns in rowData containing continuous variables. Columns containing categorical or non-atomic variables will generally be ignored.

For each DE statistic, if any `fields` are registered in `se`, they will be used directly and `patterns` will be ignored.

The registered `patterns` are used for partial name matching to the names of appropriate columns of `rowData`. All partial matching must be exact - regular expressions are not supported. Matches can occur anywhere in the name. For example, with `"PValue"`, columns with the names `"PValue.X"` and `"X.PValue"` will be considered acceptable matches. If no `patterns` are supplied, the Panels will use the values in `defaults`.

Value

All `register` functions will return `se`, modified to contain the supplied `patterns` or `fields`. These will be used as suggestions by DE-related Panels to identify the relevant fields.

All `get` functions will return a character vector containing the value set by the corresponding `register` function; or `NULL`, if nothing was set.

Author(s)

Aaron Lun

Examples

```
# Making up some results with unusual names.
se <- SummarizedExperiment(matrix(rnorm(10000), 1000, 10))
rownames(se) <- paste0("GENE_", seq_len(nrow(se)))
rowData(se)$pvalue <- runif(nrow(se))
rowData(se)$lfc <- rnorm(nrow(se))
rowData(se)$average <- rnorm(nrow(se))

se <- registerPValueFields(se, "pvalue")
getPValueFields(se)
se <- registerAveAbFields(se, "average")
getAveAbFields(se)
se <- registerLogFCFields(se, "lfc")
getLogFCFields(se)

if (interactive()) {
    iSEE(se, initial=list(MAPlot()))
}
```

`registerFeatureSetCollections` *Register feature set collections*

Description

Register feature set collations and their annotations for display in FeatureSetTables.

Usage

```
registerFeatureSetCollections(se, collections)

registerFeatureSetCommands(se, commands)

getFeatureSetCollections(se)

getFeatureSetCommands(se)
```

Arguments

se	The SummarizedExperiment object to be used in iSEE.
collections	A named list containing one or more CharacterList objects. Each entry represents a collection of feature sets (see Details) and should be named.
commands	A named list containing two character vectors of commands to use to generate collections and sets.

Details

Arbitrary feature sets are challenging as there is no obvious place to store them. registerFeatureSetCollections and friends will insert these sets into the metadata of the SummarizedExperiment object, allowing the corresponding getter functions to quickly extract them later within the iSEE app.

collections should be a named list containing CharacterList objects. Each CharacterList represents a collection where each entry is a feature set, i.e., a character vector corresponding to some of the row names of se. The mcols can contain additional per-set fields (e.g., descriptions, enrichment statistics) that will be shown in the FeatureSetTable.

commands should be a list containing:

- collections, a named character vector where each entry is named after a feature set collection. Each entry should be a string containing R commands to define a data.frame named tab, where each row is a feature set and the row names are the names of those sets.
- sets, a character vector where each entry is named after a feature set collection in the same order as commands$collections. Each entry should be a string containing R commands to define a character vector named selected containing the identity of all rows of the SummarizedExperiment in the set of interest. (These commands can assume that a .set_id variable is present containing the name of the chosen feature set, as well as the se variable containing the input SummarizedExperiment object.)

If neither collections nor commands are provided, any previously registered content in se is removed.

Value

For registerFeatureSetCollections and registerFeatureSetCommands, a modified se is returned that contains the feature set collections or commands, respectively. This can be used with FeatureSetTables in iSEE calls.

For getFeatureSetCollections, the list of CharacterLists is returned. Alternatively NULL, if no such list was stored by registerFeatureSetCollections.

For `getFeatureSetCommands`, the list of of commands is returned containing `collections` and `sets`. Alternatively `NULL`, if no such list was stored by `registerFeatureSetCommands`.

Author(s)

Aaron Lun

Examples

```
library(scRNAseq)
sce <- LunSpikeInData(location=FALSE)

# Make up some random collections.
random <- CharacterList(
   Aaron = sample(rownames(sce), 10),
   Kevin = sample(rownames(sce), 20),
   Charlotte = sample(rownames(sce), 30),
   Fed = sample(rownames(sce), 40)
)
mcols(random)$p.value <- runif(4)

# Storing the collections inside our SummarizedExperiment.
sce <- registerFeatureSetCollections(sce, list(random=random))
getFeatureSetCollections(sce)

if (interactive()) {
    iSEE(sce, initial=list(FeatureSetTable()))
}
```

---

setFeatureSetCommands *Global feature set commands*

Description

Set the commands to define the global collection of feature sets. This is deprecated in favor of `registerFeatureSetCommands`.

Usage

```
setFeatureSetCommands(value)
```

Arguments

`value`	A list of two character vectors named `"collections"` and `"sets"`. Both vectors should be of the same length and have the same names. Vectors should contain R commands to create collections and retrieve sets; see `?FeatureSetTable` and the output of `createGeneSetCommands` for details.

Value

setFeatureSetCommands will set the current global feature set commands and return NULL invisibly.

Author(s)

Aaron Lun

See Also

createGeneSetCommands, for one method of generating value.

Examples

```
old <- getFeatureSetCommands()

new.cmds <- createGeneSetCommands(organism="org.Mm.eg.db",
    identifier="SYMBOL")
setFeatureSetCommands(new.cmds)

getFeatureSetCommands()

setFeatureSetCommands(old)
```

---

utils-geneset *Gene set utilities*

Description

Utility functions to control the behavior of the GeneSetTable.

Usage

```
.getIdentifierType()

.setIdentifierType(value)

.getOrganism()

.setOrganism(value)

.getGeneSetCommands(collection, mode)

.setGeneSetCommands(value)
```

Arguments

value	For .setIdentifierType and .setOrganism, a string containing the type of identifier or organism package to use. For .setGeneSetCommands, a named list containing two character vectors, see Details.
collection	String specifying the gene set collection.
mode	String specifying the mode of operation for the returned commands.

Details

By default, .getGeneSetCommands will extract GO and KEGG terms. The organism and identifier type relates to the manner in which this default extraction is performed.

Users can add their own gene set collections by supplying a named list to .setGeneSetCommands. Each element of the list should be a named character vector of length two, with names "show" and "extract" - see the return value for what these are. The names of the list should be unique and will be used in the GeneSetTable interface.

Alternatively, any element of the list may be NULL, in which case it is excluded from the interface. This is useful for setting, e.g., GO=NULL to ignore the in-built GO terms.

Value

.getIdentifierType will return the identifier type to use, defaulting to "ENTREZID".

.getOrganism will return the organism package to use, defaulting "org.Hs.eg.db".

.getGeneSetCommands will return:

- If mode="show", a string containing R commands that create tab, a data.frame of all gene sets for a given collection.
- If mode="extract", a format string containing R commands that (after formatting) create selected, a character vector of gene identities for the selected gene set. This format string should accept one string argument corresponding to the deparsed name of the gene set.

Each of the setter functions will set the corresponding option and return NULL, invisibly.

Author(s)

Aaron Lun

See Also

GeneSetTable, where these functions have their effect.

Examples

```
.setIdentifierType("ENSEMBLID")
.getIdentifierType()

.setOrganism("org.Mm.eg.db")
.getOrganism()
```

```
.getGeneSetCommands("GO", "show")
.getGeneSetCommands("GO", "extract")

.setGeneSetCommands(
    list(AaronRandomCollection=
        c(
            show='tab <- some_function_to_list_my_gene_sets()',
            extract='selected <- some_function_to_get_one_gene_set(%s)'
        )
    )
)

.getGeneSetCommands("AaronRandomCollection", "show")
.getGeneSetCommands("AaronRandomCollection", "extract")
```

`VolcanoPlot-class` *The VolcanoPlot class*

Description

The VolcanoPlot is a RowDataPlot subclass that is dedicated to creating a volcano plot. It retrieves the log-fold change and p-value from and creates a row-based plot where each point represents a feature.

Slot overview

The following slots control the thresholds used in the visualization:

- `PValueThreshold`, a numeric scalar in (0, 1] specifying the threshold to use on the (adjusted) p-value. Defaults to 0.05.
- `LogFCThreshold`, a non-negative numeric scalar specifying the threshold to use on the log-fold change. Defaults to 0.
- `PValueCorrection`, a string specifying the multiple testing correction to apply. Defaults to `"BH"`, but can take any value from `p.adjust.methods`.

In addition, this class inherits all slots from its parent RowDataPlot, RowDotPlot, DotPlot and Panel classes.

Constructor

`VolcanoPlot(...)` creates an instance of a VolcanoPlot class, where any slot and its value can be passed to `...` as a named argument.

Users are expected to load relevant statistics into the `rowData` of a SummarizedExperiment. This panel expects one or more columns containing the p-values and log-fold changes for each gene/row - see Examples. The expected column names (and how to tune them) are listed at `?"registerPValueFields"`.

Supported methods

In the following code snippets, x is an instance of a RowDataPlot class. Refer to the documentation for each method for more details on the remaining arguments.

For setting up data values:

- .cacheCommonInfo(x, se) returns se after being loaded with class-specific constants. This includes "valid.p.fields" and "valid.lfc.fields", character vectors containing the names of valid rowData columns for the p-values and log-fold changes, respectively.
- .refineParameters(x, se) returns x after setting XAxis="Row data" and the various *Pattern fields to their cached values. This will also call the equivalent RowDataPlot method for further refinements to x. If valid p-value and log-fold change fields are not available, NULL is returned instead.

For defining the interface:

- .defineDataInterface(x, se, select_info) returns a list of interface elements for manipulating all slots described above.
- .panelColor(x) will return the specified default color for this panel class.
- .allowableXAxisChoices(x, se) returns a character vector specifying the acceptable log-fold change-related variables in rowData(se) that can be used as choices for the x-axis.
- .allowableYAxisChoices(x, se) returns a character vector specifying the acceptable p-value-related variables in rowData(se) that can be used as choices for the y-axis.
- .hideInterface(x, field) will return TRUE for field="XAxis", otherwise it will call the RowDataPlot method.
- .fullName(x) will return "Volcano plot".

For monitoring reactive expressions:

- .createObservers(x, se, input, session, pObjects, rObjects) sets up observers for all new slots described above, as well as in the parent classes via the RowDataPlot method.

For creating the plot:

- .generateDotPlotData(x, envir) will create a data.frame of row metadata variables in envir. This should contain negative log-transformed p-values on the y-axis and log-fold changes on the x-axis, in addition to an extra field specifying whether or not the feature was considered to be significantly up or down. The method will return the commands required to do so as well as a list of labels.
- .prioritizeDotPlotData(x, envir) will create variables in envir marking the priority of points. Significant features receive higher priority (i.e., are plotted over their non-significant counterparts) and are less aggressively downsampled when Downsample=TRUE. The method will return the commands required to do this as well as a logical scalar indicating that rescaling of downsampling resolution is performed.
- .colorByNoneDotPlotField(x) will return a string specifying the field of the data.frame (generated by .generateDotPlotData) containing the significance information. This is to be used for coloring when ColorBy="None".
- .colorByNoneDotPlotScale(x) will return a string containing a **ggplot2** command to add a default color scale when ColorBy="None".

- `.generateDotPlot(x, labels, envir)` returns a list containing `plot` and `commands`, using the inital ColumnDataPlot ggplot and adding vertical lines demarcating the log-fold change threshold.

For documentation:

- `.definePanelTour(x)` returns an data.frame containing the steps of a panel-specific tour.
- `.getDotPlotColorHelp(x, color_choices)` returns a function that generates an **rintrojs** tour for the color choice UI.

Author(s)

Aaron Lun

See Also

RowDataPlot, for the base class.

Examples

```
# Making up some results:
se <- SummarizedExperiment(matrix(rnorm(10000), 1000, 10))
rownames(se) <- paste0("GENE_", seq_len(nrow(se)))
rowData(se)$PValue <- runif(nrow(se))
rowData(se)$LogFC <- rnorm(nrow(se))
rowData(se)$AveExpr <- rnorm(nrow(se))

if (interactive()) {
    iSEE(se, initial=list(VolcanoPlot()))
}
```

Index