Package 'iSEE'

October 16, 2019

Title Interactive SummarizedExperiment Explorer

Version 1.4.0

Date 2019-04-13

Description Provides functions for creating an interactive Shiny-based graphical user interface for exploring data stored in SummarizedExperiment objects, including row- and column-level metadata. Particular attention is given to single-cell data in a SingleCellExperiment object with visualization of dimensionality reduction results.

Depends SummarizedExperiment, SingleCellExperiment

Imports methods, BiocGenerics, S4Vectors, utils, stats, shiny, shinydashboard, shinyAce, shinyjs, DT, rintrojs, ggplot2, colourpicker, igraph, vipor, mgcv, reshape2, rentrez, AnnotationDbi, graphics, grDevices, viridisLite, cowplot, scales, dplyr

Suggests testthat, BiocStyle, knitr, rmarkdown, scRNAseq, scater, DelayedArray (>= 0.7.44), Rtsne, irlba, RColorBrewer, viridis, org.Mm.eg.db, htmltools

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

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

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

License MIT + file LICENSE

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.1.1

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

git_branch RELEASE_3_9

git_last_commit 6af7a42

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

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

Maintainer Charlotte Soneson <charlottesoneson@gmail.com>

R topics documented:

`annotateEnsembl` *Annotation via ENSEMBL database*

Description

Annotation facility for displaying additional information on selected genes, based on the data retrieved from the ENSEMBL database

Usage

```
annotateEnsembl(se, orgdb, keytype, rowdata_col = NULL,
  ens_species = gsub(" ", "_", species(orgdb)))
```

Arguments

se	An object that is coercible to SingleCellExperiment.
orgdb	An OrgDb object, as a basis for the annotation. Typical values are org.Hs.eg.db for human, org.Mm.eg.db for mouse, and so on. The corresponding package has to be available and loaded before calling this function
keytype	The keytype that matches the IDs used in the se object. Can be one of the values of keytypes(org.XX.eg.db), typically being "SYMBOL", "ENSEMBL", or "ENTREZID".
rowdata_col	A character string specifying which column of rowData(se) contains the keys. Defaults to NULL, which corresponds to having the ids of the features as row names of the se object itself.
ens_species	Character string containing the species name, coded as in the ENSEMBL database and browser. For example, "Homo_sapiens" for human and "Mus_musculus". Defaults to species(orgdb), with the whitespace replaced by underscore

Value

A function to be used as the value of the annotFun parameter of iSEE. This function itself returns a HTML tag object with the content extracted from the call, accepting as parameters the se object and the row_index corresponding to the feature of interest.

Examples

```
library(scRNAseq)
data(allen)
sce <- as(allen, "SingleCellExperiment")
library(org.Mm.eg.db)
myfun <- annotateEnsembl(sce, org.Mm.eg.db, keytype="SYMBOL")
## Not run:
# Requires a working internet connection
myfun(sce, 4242)

## End(Not run)

# to be used when launching the app itself ----

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

---

annotateEntrez *Annotation via Entrez database*

Description

Annotation facility for displaying additional information on selected genes, based on the data retrieved from the ENTREZ database

Usage

```
annotateEntrez(se, orgdb, keytype, rowdata_col = NULL)
```

Arguments

se	An object that is coercible to SingleCellExperiment.
orgdb	An OrgDb object, as a basis for the annotation. Typical values are org.Hs.eg.db for human, org.Mm.eg.db for mouse, and so on. The corresponding package has to be available and loaded before calling this function.
keytype	The keytype that matches the IDs used in the se object. Can be one of keytypes(org.XX.eg.db), typically "SYMBOL", "ENSEMBL", or "ENTREZID".
rowdata_col	A character string specifying which column of rowData(se) contains the keys. Defaults to NULL, which corresponds to having the ids of the features as row names of the se object itself.

Value

A function to be used as the value of the annotFun parameter of iSEE. This function itself returns a HTML tag object with the content extracted from the call, accepting as parameters the se object and the row_index corresponding to the feature of interest.

Examples

```
library(scRNAseq)
data(allen)
sce <- as(allen, "SingleCellExperiment")

library(org.Mm.eg.db)
myfun <- annotateEntrez(sce, org.Mm.eg.db, keytype="SYMBOL")
## Not run:
# Requires a working internet connection
myfun(sce, 4242)

## End(Not run)

# to be used when launching the app itself ----

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

---

colDataPlotDefaults *Column data plot defaults*

Description

Create default settings for column data plot panels in the iSEE interface.

Usage

```
colDataPlotDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of column data plots that can be added to the interface.

Details

Parameters available to column data plots are:

YAxis: Character, which column of `colData(se)` should be shown on the y-axis? Defaults to the first entry of `colData(se)`.

XAxis: Character, what type of variable should be shown on the x-axis? Defaults to `"None"`, but can also be `"Column data"`.

XAxisColData: Character, which column of `colData(se)` should be shown on the x-axis if `XAxis="Column data"`? Defaults to the first entry of `colData(se)`.

All column-based parameters described in `?"iSEE point parameters"` are applicable. All plot-based parameters described in `?"iSEE selection parameters"` are applicable.

Value

A DataFrame containing default settings for parameters of each of `number` column data panels.

Author(s)

Aaron Lun

See Also

`?"iSEE point parameters"`, `?"iSEE selection parameters"`

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
colDataPlotDefaults(sce, n=1)
```

---

colStatTableDefaults *Column statistics table defaults*

Description

Create default settings for column statistics table panels in the iSEE interface.

Usage

```
colStatTableDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of column statistics tables that can be added to the interface.

Details

Parameters available to col statistics tables are:

`Selected`: Integer, containing the index of the col to be initially selected. Defaults to the first col, i.e., 1. Alternatively, a string can be supplied containing the column name.

`Search`: Character, containing the initial value of the search field. Defaults to an empty string.

`SearchColumns`: A list containing character vectors of length equal to the number of columns in `colData(se)`, specifying the initial value of the search field for each column. All entries default to an empty string.

All table-based parameters described in ?"iSEE selection parameters" are applicable.

Value

A DataFrame containing default settings for parameters of each of `number` column statistics table panels.

Author(s)

Aaron Lun

See Also

?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
colStatTableDefaults(sce, n=1)
```

---

customDataPlotDefaults

Custom data plot defaults

Description

Create default settings for custom data plot panels in the iSEE interface.

Usage

```
customDataPlotDefaults(se, number)
```

Arguments

`se`	A SummarizedExperiment object.
`number`	An integer scalar, specifying the maximum number of custom data plots that can be added to the interface.

Details

Data parameters available to custom data plots are:

`DataBoxOpen`: Logical, should the data parameter box be open upon initialization? Defaults to `FALSE`.

`Function`: String, which function should be used to generate the ggplot for the curent panel? Defaults to `"---"`, i.e., no coordinates are generated.

`Arguments`: String with multiple lines specifying the initial arguments for the function, see `vignette("custom",packag` for details. Defaults to an empty string.

`VisibleArgs`: String with multiple lines specifying the initial arguments to be shown in the text area. This can differ from `Arguments`, e.g., to list argument names without their values for users to enter. Defaults to `NA`, which means that the value in `Arguments` will be shown.

Selection parameters for custom data plots are:

`SelectBoxOpen`: Logical, should the selection parameter box be open upon initialization? Defaults to `FALSE`.

`ColumnSource`: Character, which other plot should transmit sample selections to the current plot? Defaults to `"---"`, which means that no plot is used for point selection.

`RowSource`: Character, which other plot should transmit feature selections to the current plot? Defaults to `"---"`, which means that no plot is used for point selection.

Value

A DataFrame containing default settings for parameters of each of `number` custom data plot panels.

Author(s)

Aaron Lun, Kevin Rue-Albrecht

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
customDataPlotDefaults(sce, n=1)
```

---

customStatTableDefaults

Custom statistics table defaults

Description

Create default settings for custom statistics table panels in the iSEE interface.

Usage

```
customStatTableDefaults(se, number)
```

Arguments

`se`	A SummarizedExperiment object.
`number`	An integer scalar, specifying the maximum number of custom statistics tables that can be added to the interface.

Details

Data parameters available to custom statistics tables are:

`DataBoxOpen`: Logical, should the data parameter box be open upon initialization? Defaults to `FALSE`.

`Function`: String, which function should be used to generate a data.frame? Defaults to `"---"`, i.e., no coordinates are generated.

`Arguments`: String with multiple lines specifying the initial arguments for the function, see `vignette("custom",packag` for details. Defaults to an empty string.

`VisibleArgs`: String with multiple lines specifying the initial arguments to be shown in the text area. This can differ from `Arguments`, e.g., to list argument names without their values for users to enter. Defaults to `NA`, which means that the value in `Arguments` will be shown.

Selection parameters for custom statistics tables are:

`SelectBoxOpen`: Logical, should the selection parameter box be open upon initialization? Defaults to `FALSE`.

`ColumnSource`: Character, which other plot should transmit sample selections to the current table? Defaults to `"---"`, which means that no plot is used for point selection.

`RowSource`: Character, which other plot should transmit feature selections to the current table? Defaults to `"---"`, which means that no plot is used for point selection.

Tables also have an `Search` field indicating what string should be put into the search box. This defaults to an empty string.

Value

A DataFrame containing default settings for parameters of each of `number` custom statistics table panels.

Author(s)

Aaron Lun, Kevin Rue-Albrecht

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
customStatTableDefaults(sce, n=1)
```

ExperimentColorMap-class

ExperimentColorMap class

Description

ExperimentColorMap class

Usage

```
ExperimentColorMap(assays = list(), colData = list(),
  rowData = list(), all_discrete = list(assays = NULL, colData = NULL,
  rowData = NULL), all_continuous = list(assays = NULL, colData = NULL,
  rowData = NULL), global_discrete = NULL, global_continuous = NULL,
  ...)
```

Arguments

assays	List of colormaps for assays.
colData	List of colormaps for colData.
rowData	List of colormaps for rowData.
all_discrete	Colormaps applied to all undefined categorical assays, colData, and rowData, respectively.
all_continuous	Colormaps applied to all undefined continuous assays, colData, and rowData, respectively.
global_discrete	Colormap applied to all undefined categorical covariates.
global_continuous	Colormap applied to all undefined continuous covariates.
...	additional arguments passed on to the ExperimentColorMap constructor

Details

Colormaps must all be functions that take at least one argument: the number of (named) colours to return as a character vector. This argument may be ignored in the body of the colormap function to produce constant colormaps.

Value

An object of class ExperimentColorMap

Categorical colormaps

The default categorical colormap emulates the default ggplot2 categorical color palette (Credit: https://stackoverflow.com/questions/8197559/emulate-ggplot2-default-color-palette). This palette returns a set of colors sampled in steps of equal size that correspond to approximately equal perceptual changes in color:

```
function(n) {
    hues=seq(15, 375, length=(n + 1))
    hcl(h=hues, l=65, c=100)[seq_len(n)]
}
```

To change the palette for all categorical variables, users must supply a colormap that returns a similar value; namely, an unnamed character vector of length n. For instance, using the base R palette rainbow.colors

```
function(n) {
    rainbow(n)
}
```

Accessors

In the following code snippets, `x` is an `ExperimentColorMap` object. If the colormap can not immediately be found in the appropriate slot, `discrete` is a `logical(1)` that indicates whether the default colormap returned should be categorical `TRUE` or continuous (`FALSE`, default).

`assayColorMap(x, i, ..., discrete=FALSE)`**:** Get an `assays` colormap.

`colDataColorMap(x, i, ..., discrete=FALSE)`**:** Get a `colData` colormap.

`rowDataColorMap(x, i, ..., discrete=FALSE)`**:** Get a `rowData` colormap.

Setters

In the following code snippets, `x` is an `ExperimentColorMap` object, and `i` is a character or numeric index.

`assayColorMap(x, i, ...) <- value`**:** Set an `assays` colormap.

`colDataColorMap(x, i, ...) <- value`**:** Set a `colData` colormap.

`rowDataColorMap(x, i, ...) <- value`**:** Set a `rowData` colormap.

`assay(x, i, ...) <- value`**:** Alias. Set an `assays` colormap.

Examples

```
# Example colormaps ----

count_colors <- function(n){
  c("black", "brown", "red", "orange", "yellow")
}
fpkm_colors <- viridis::inferno
tpm_colors <- viridis::plasma

qc_color_fun <- function(n){
  qc_colors <- c("forestgreen", "firebrick1")
  names(qc_colors) <- c("Y", "N")
  return(qc_colors)
}

# Constructor ----

ecm <- ExperimentColorMap(
    assays=list(
        counts=count_colors,
        tophat_counts=count_colors,
        cufflinks_fpkm=fpkm_colors,
        rsem_tpm=tpm_colors
    ),
    colData=list(
        passes_qc_checks_s=qc_color_fun
    )
)

# Accessors ----

# assay colormaps
assayColorMap(ecm, "logcounts") # [undefined --> default]
```

```
assayColorMap(ecm, "counts")
assayColorMap(ecm, "cufflinks_fpkm")
assay(ecm, "cufflinks_fpkm") # alias

# colData colormaps
colDataColorMap(ecm, "passes_qc_checks_s")
colDataColorMap(ecm, "undefined")

# rowData colormaps
rowDataColorMap(ecm, "undefined")

# generic accessors
assays(ecm)
assayNames(ecm)

# Setters ----

assayColorMap(ecm, "counts") <- function(n){c("blue", "white", "red")}
assay(ecm, 1) <- function(n){c("blue", "white", "red")}

colDataColorMap(ecm, "passes_qc_checks_s") <- function(n){NULL}
rowDataColorMap(ecm, "undefined") <- function(n){NULL}

# Categorical colormaps ----

# Override all discrete colormaps using the base rainbow palette
ecm <- ExperimentColorMap(global_discrete = rainbow)
n <- 10
plot(1:n, col=assayColorMap(ecm, "undefined", discrete = TRUE)(n), pch=20, cex=3)
```

`featAssayPlotDefaults` *Feature assay plot defaults*

Description

Create default settings for feature assay plot panels in the iSEE interface.

Usage

```
featAssayPlotDefaults(se, number)
```

Arguments

`se`	A SummarizedExperiment object.
`number`	An integer scalar, specifying the maximum number of feature assay plots that can be added to the interface.

Details

Parameters available to feature assay plots are:

`YAxisFeatName`**:** Integer, the index of the feature for which to show the expression on the y-axis if `YAxis="Feature name"`. Defaults to 1, i.e., the first feature in `se`. Alternatively, a string can be supplied containing the name of the feature, i.e., the row name.

`YAxisRowTable`: Character, what row statistics table should be used to choose a feature to display on the y-axis? Any setting will override `YAxisFeatName` with the selected row in the chosen table upon initialization of the app. Defaults to `"---"`, which means that no table will be used.

`Assay`: Integer, which assay should be used to supply the expression values shown on the y-axis? Defaults to 1, i.e., the first assay in `se`. Alternatively, a string can also be supplied containing the name of the assay, if `assays(se)` has names.

`XAxis`: Character, what type of variable should be shown on the x-axis? Defaults to `"None"`, but can also be `"Row table"`, `"Column data"` or `"Feature name"`.

`XAxisColData`: Character, what column of `colData(se)` should be shown on the x-axis if `XAxis="Column data"`? Defaults to the first entry of `colData(se)`.

`XAxisFeatName`: Integer, the index of the feature for which to show the expression on the x-axis if `XAxis="Feature name"`. Defaults to 1, i.e., the first feature in `se`. Alternatively, a string can be supplied containing the name of the feature.

`XAxisRowTable`: Character, which row statistic table should be used to choose a feature to put on the x-axis if `XAxis="Row table"`? Any setting will override `XAxisFeatName` with the selected row in the chosen table upon initialization of the app. Defaults to `"---"`, which means that no table will be used.

All column-based parameters described in `?"iSEE point parameters"` are applicable. All plot-based parameters described in `?"iSEE selection parameters"` are applicable.

Value

A DataFrame containing default settings for parameters of each of `number` feature assay panels.

Author(s)

Aaron Lun

See Also

`?"iSEE point parameters"`, `?"iSEE selection parameters"`

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
featAssayPlotDefaults(sce, n=1)
```

---

heatMapPlotDefaults *Heatmap defaults*

Description

Create default settings for heatmap panels in the iSEE interface.

Usage

```
heatMapPlotDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of heatmaps that can be added to the interface.

Details

The features/rows to be used in the construction of the heatmap are specified with:

FeatName: List of length equal to the number of panels. Each list entry corresponds to a panel and should be an integer vector with the indices of the feature(s) for which to show the expression in the heatmap. Defaults to 1L for each panel, i.e., the first feature in se. Alternatively, a character vector can be supplied containing the names of the features.

Assay: Integer, which assay should be used to supply the expression values shown on the y-axis? Defaults to 1, i.e., the first assay in se. Alternatively, a string can also be supplied containing the name of the assay, if assays(se) has names.

FeatNameSource: Character, which other panel should be used to choose the features to show in the heatmap? Defaults to "---", which means that no panel is used for feature selection.

FeatNameBoxOpen: Logical, should the feature selection box be open upon initialization? Defaults to FALSE.

The column metadata variables control the ordering of the samples in the heatmap. They can be controlled with:

ColData: List of length equal to the number of panels. Each list entry corresponds to a panel and should contain a character vector specifying the field(s) of colData(se) that should be used to order the samples in the heatmap. Note that these fields will also appear as annotation bars. Each character vector defaults to the first entry of colData(se).

ColDataBoxOpen: Logical, should the column data selection box be open upon initialization? Defaults to FALSE.

A variety of parameters are available to control the color scale of the heatmap. They can be specified with:

ColorBoxOpen: Logical, should the color selection panel for the heatmap be open upon initialization? Defaults to FALSE.

CenterScale: List of length equal to the number of panels. Each list entry corresponds to a panel and contains a character vector specifying whether each row of expression values should be mean-centered and/or scaled to unit variance. Defaults to "Centered". Users can set it to c("Centered","Scaled") to obtain mean-centered and unit-scaled rows.

Lower: Numeric, what should be the lower bound of the color scale for the values in the heatmap? All values below this threshold will be shown in the same color. Defaults to -Inf, meaning that the lowest value in the data matrix will be used.

Upper: Numeric, what should be the upper bound of the color scale for the values in the heatmap? All values above this threshold will be shown in the same color. Defaults to Inf, meaning that the highest value in the data matrix will be used.

ColorScale: Character, what color scale (in the form low-mid-high) should be used to color the heatmap when values are centered? Defaults to "purple-black-yellow".

The ZoomData field for heatmaps should contain an integer vector of consecutive indices to zoom into from the full heatmap. This vector will subset the entries in FeatName for a given panel. This defaults to NULL, i.e., all specified features in FeatName are shown.

All plot-based parameters described in ?"iSEE selection parameters" are also applicable.

Value

A DataFrame containing default settings for parameters of each of number heatmap panels.

Author(s)

Charlotte Soneson

See Also

?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
heatMapPlotDefaults(sce, n=1)
```

---

isColorMapCompatible *Check compatibility between ExperimentColorMap and SummarizedExperiment objects*

Description

This function compares a pair of ExperimentColorMap and SingleCellExperiment objects, and examines whether all of the assays, colData, and rowData defined in the ExperimentColorMap object exist in the SingleCellExperiment object.

Usage

```
isColorMapCompatible(ecm, se, error = FALSE)
```

Arguments

ecm	An ExperimentColorMap.
se	A SingleCellExperiment.
error	A logical value that indicates whether an informative error should be thrown, describing why the two objects are not compatible.

Value

A logical value that indicates whether a given pair of ExperimentColorMap and SummarizedExperiment objects are compatible. If error=TRUE, an informative error is thrown, rather than returing FALSE.

Author(s)

Kevin Rue-Albrecht

Examples

```
# Example colormaps ----

count_colors <- function(n){
  c("black","brown","red","orange","yellow")
}

qc_color_fun <- function(n){
  qc_colors <- c("forestgreen", "firebrick1")
  names(qc_colors) <- c("Y", "N")
  return(qc_colors)
}

ecm <- ExperimentColorMap(
    assays = list(
        tophat_counts = count_colors
    ),
    colData = list(
        passes_qc_checks_s = qc_color_fun
    )
)

# Example SingleCellExperiment ----

library(scRNAseq)
data(allen)
library(scater)
sce <- as(allen, "SingleCellExperiment")

# Test for compatibility ----

isColorMapCompatible(ecm, sce)
```

---

iSEE *iSEE: interactive SummarizedExperiment/SingleCellExperiment Explorer*

Description

Interactive and reproducible visualization of data contained in a SummarizedExperiment/SingleCellExperiment, using a Shiny interface.

Usage

```
iSEE(se, redDimArgs = NULL, colDataArgs = NULL, featAssayArgs = NULL,
  rowStatArgs = NULL, rowDataArgs = NULL, sampAssayArgs = NULL,
  colStatArgs = NULL, customDataArgs = NULL, customStatArgs = NULL,
  heatMapArgs = NULL, redDimMax = 5, colDataMax = 5,
  featAssayMax = 5, rowStatMax = 5, rowDataMax = 5,
  sampAssayMax = 5, colStatMax = 5, customDataMax = 5,
  customStatMax = 5, heatMapMax = 5, initialPanels = NULL,
```

```
annotFun = NULL, customDataFun = NULL, customStatFun = NULL,
customSendAll = FALSE, colormap = ExperimentColorMap(),
tour = NULL, appTitle = NULL, runLocal = TRUE, voice = FALSE)
```

Arguments

se	An object that is coercible to SingleCellExperiment.
redDimArgs	A DataFrame similar to that produced by redDimPlotDefaults, specifying initial parameters for the reduced dimension plots.
colDataArgs	A DataFrame similar to that produced by colDataPlotDefaults, specifying initial parameters for the column data plots.
featAssayArgs	A DataFrame similar to that produced by featAssayPlotDefaults, specifying initial parameters for the feature assay plots.
rowStatArgs	A DataFrame similar to that produced by rowStatTableDefaults, specifying initial parameters for the row statistics tables.
rowDataArgs	A DataFrame similar to that produced by rowDataPlotDefaults, specifying initial parameters for the row data plots.
sampAssayArgs	A DataFrame similar to that produced by sampAssayPlotDefaults, specifying initial parameters for the sample assay plots.
colStatArgs	A DataFrame similar to that produced by colStatTableDefaults, specifying initial parameters for the sample assay plots.
customDataArgs	A DataFrame similar to that produced by customDataPlotDefaults, specifying initial parameters for the custom data plots.
customStatArgs	A DataFrame similar to that produced by customStatTableDefaults, specifying initial parameters for the custom statistics tables.
heatMapArgs	A DataFrame similar to that produced by heatMapPlotDefaults, specifying initial parameters for the heatmaps.
redDimMax	An integer scalar specifying the maximum number of reduced dimension plots in the interface.
colDataMax	An integer scalar specifying the maximum number of column data plots in the interface.
featAssayMax	An integer scalar specifying the maximum number of feature assay plots in the interface.
rowStatMax	An integer scalar specifying the maximum number of row statistics tables in the interface.
rowDataMax	An integer scalar specifying the maximum number of row data plots in the interface.
sampAssayMax	An integer scalar specifying the maximum number of sample assay plots in the interface.
colStatMax	An integer scalar specifying the maximum number of column statistics tables in the interface.
customDataMax	An integer scalar specifying the maximum number of custom data plots in the interface.
customStatMax	An integer scalar specifying the maximum number of custom statistics tables in the interface.
heatMapMax	An integer scalar specifying the maximum number of heatmaps in the interface.

initialPanels	A DataFrame specifying which panels should be created at initialization. This should contain a Name character field and may have optional Width and Height integer fields, see Details.
annotFun	A function, similar to those returned by annotateEntrez or annotateEnsembl. The function should accept two parameters, se and row_index, and return a HTML element with annotation for the selected row.
customDataFun	A named list of functions for reporting coordinates to use in a custom data plot.
customStatFun	A named list of functions for reporting coordinates to use in a custom statistics table.
customSendAll	A logical scalar indicating whether all (active and saved) selections should be passed from transmitting panels to custom plots or tables. The default (FALSE) only passes the row names of the points in the active selection.
colormap	An ExperimentColorMap object that defines custom colormaps to apply to individual assays, colData and rowData covariates.
tour	A data.frame with the content of the interactive tour to be displayed after starting up the app.
appTitle	A string indicating the title to be displayed in the app. If not provided, the app displays the version info of iSEE.
runLocal	A logical indicating whether the app is to be run locally or remotely on a server, which determines how documentation will be accessed.
voice	A logical indicating whether the voice recognition should be enabled.

Details

Users can pass default parameters via DataFrame objects in redDimArgs and featAssayArgs. Each object can contain some or all of the expected fields (see redDimPlotDefaults). Any missing fields will be filled in with the defaults.

The number of maximum plots for each type of plot is set to the larger of *Max and nrow(*Args). Users can specify any number of maximum plots, though increasing the number will increase the time required to render the interface.

The initialPanels argument specifies the panels to be created upon initializing the interface. This should be a DataFrame containing a Name field specifying the identity of the panel, e.g., "Reduced dimension plot 1", "Row statistics table 2". Please refer to availablePanelTypes for the full list of panels available. The trailing number should not be greater than the number of maximum plots of that type. Users can also define the Width field, specifying the width of each panel from 2 to 12 (values will be coerced inside this range); and the Height field, specifying the height of each panel from 400 to 1000 pixels. By default, one panel of each type (where possible) will be generated, with height of 500 and width of 4.

The tour argument needs to be provided in a form compatible with the format expected by the rintrojs package. There should be two columns, element and intro, with the former describing the element to highlight and the latter providing some descriptive text. See https://github.com/carlganz/rintrojs#usage for more information.

By default, categorical data types such as factor and character are limited to 24 levels, beyond which they are coerced to numeric variables for faster plotting. This limit may be set to a different value as a global option, e.g. options(iSEE.maxlevels=30).

Value

A Shiny app object is returned, for interactive data exploration of the SummarizedExperiment or SingleCellExperiment object.

Examples

```
library(scRNAseq)
data(allen)
class(allen)

# Example data ----

library(scater)
sce <- as(allen, "SingleCellExperiment")
counts(sce) <- assay(sce, "tophat_counts")
sce <- normalize(sce)

sce <- runPCA(sce, ncomponents=4)
sce <- runTSNE(sce)
rowData(sce)$ave_count <- rowMeans(counts(sce))
rowData(sce)$n_cells <- rowSums(counts(sce)>0)
sce

# launch the app itself ----

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

`iSEE point parameters` *Point plot aesthetic parameters*

Description

Parameters related to aesthetics for point-based plotting panels.

Coloring parameters

`ColorBy`**:** Character, what type of data should be used for coloring? Defaults to `"None"`, but can also be `"Feature name"`, `"Sample name"`, or `"Column data"` (for column-based plots) or `"Row data"` for (row-based plots).

`ColorByDefaultColor`**:** String specifying the default point colour when `ColorBy="None"`. Defaults to `"black"`.

`ColorByFeatName`**:** Integer, the index of the feature to use if `ColorBy="Feature name"`. Defaults to 1, i.e., the first feature in `se`. Alternatively, a string can be supplied containing the name of the feature.

`ColorBySampName`**:** Integer, the index of the sample to use if `ColorBy="Sample name"`. Defaults to 1, i.e., the first sample in `se`. Alternatively, a string can be supplied containing the name of the sample.

`ColorByRowTable`**:** Character, which row statistics table should be used to choose a feature to color by, if `ColorBy="Feature name"`? Any setting will override `ColorByFeatName` with the selected row in the chosen table upon initialization of the app. Defaults to `"---"`, which means that no table will be used.

`ColorByColTable`: Character, which column statistics table should be used to choose a sample to color by, if `ColorBy="Sample name"`? Any setting will override `ColorBySampName` with the selected row in the chosen table upon initialization of the app. Defaults to `"---"`, which means that no table will be used.

For the plots where each point represents a sample (i.e., all plots except for heatmaps and row data plots), the following additional options apply:

`ColorByColData`: Character, which column of `colData(se)` should be used for colouring if `ColorBy="Column data"`? Defaults to the first entry of `colData(se)`.

`ColorByFeatNameAssay`: Integer, which assay should be used to supply the expression values for colouring if `ColorBy="Feature name"`? Defaults to 1, i.e., the first assay in `se`. Alternatively, a string can also be supplied containing the name of the assay, if `assays(se)` has names.

`ColorBySampNameColor`: String specifying the colour to be used to highlight the selected sample if `ColorBy="Sample name"`. Defaults to `"red"`.

For plots where each point represents a feature (i.e., row data plots), the following additional options apply:

`ColorByRowData`: Character, which column of `rowData(se)` should be used for colouring if `ColorBy="Row data"`? Defaults to the first entry of `rowData(se)`.

`ColorByFeatNameColor`: String specifying the colour to be used to highlight the selected feature if `ColorBy="Feature name"`. Defaults to `"red"`.

`ColorBySampNameAssay`: Integer, which assay should be used to supply the expression values for colouring if `ColorBy="Sample name"`? Defaults to 1, i.e., the first assay in `se`. Alternatively, a string can also be supplied containing the name of the assay, if `assays(se)` has names.

Shape parameters

`ShapeBy`: Character, what type of data should be used for controlling shape? Defaults to `"None"`, but can also be `"Column data"`.

For the plots where each point represents a sample (i.e., all plots except for heatmaps and row data plots), the following additional options apply:

`ShapeByColData`: Character, which column of `colData(se)` should be used for shaping if `ShapeBy="Column data"`? This should refer to a categorical variable, and will default to the first such entry of `colData(se)`.

For plots where each point represents a feature (i.e., row data plots), the following additional options apply:

`ShapeByRowData`: Character, which column of `rowData(se)` should be used for shaping if `ShapeBy="Row data"`? This should refer to a categorical variable, and will default to the first such entry of `rowData(se)`.

Size parameters

`SizeBy`: Character, what type of data should be used for controlling size? Defaults to `"None"`, but can also be `"Column data"`.

For the plots where each point represents a sample (i.e., all plots except for heatmaps and row data plots), the following additional options apply:

`SizeByColData`: Character, which column of `colData(se)` should be used for sizing if `SizeBy="Column data"`? This should refer to a continuous variable, and will default to the first such entry of `colData(se)`.

For plots where each point represents a feature (i.e., row data plots), the following additional options apply:

`SizeByRowData`: Character, which column of `rowData(se)` should be used for sizing if `SizeBy="Row data"`? This should refer to a continuous variable, and will default to the first such entry of `rowData(se)`.

Contour line parameters

`ContourAdd`: Logical, should contour lines be added to the plot?

`ContourColor`: String specifying the default colour of contour lines.

Faceting parameters

`FacetByRow`: Logical indicating whether the plot should be faceted by row.

`FacetByColumn`: Logical indicating whether the plot should be faceted by row.

For the plots where each point represents a sample (i.e., all plots except for heatmaps and row data plots), the following additional options apply:

`RowFacetByColData`: Character, which column of `colData(se)` should be used to facet by row? This should refer to a categorical variable, and will default to the first such entry of `colData(se)`.

`ColumnFacetByColData`: Character, which column of `colData(se)` should be used to facet by column? This should refer to a categorical variable, and will default to the first such entry of `colData(se)`.

For the plots where each point represents a feature (i.e., row data plots), the following additional options apply:

`RowFacetByRowData`: Character, which column of `colData(se)` should be used to facet by row? This should refer to a categorical variable, and will default to the first such entry of `colData(se)`.

`ColumnFacetByRowData`: Character, which column of `colData(se)` should be used to facet by column? This should refer to a categorical variable, and will default to the first such entry of `colData(se)`.

Other plot parameters

Parameter visibility options are:

`DataBoxOpen`: Logical, should the data parameter box be open upon initialization? Defaults to `FALSE`.

`VisualBoxOpen`: Logical, should the visual parameter box be open upon initialization? Defaults to `FALSE`.

`VisualChoices`: A list containing one character vector, specifying the visual box parameters to be shown upon initialization. This defaults to `list("Color")` to show only the coloring parameters, but the internal vector can also contain `"Points"`, `"Facets"` and `"Other"`.

Options related to the appearance of points are:

`PointSize`: Numeric, the size of the points. Defaults to 1.

`PointAlpha`: Numeric, what level of transparency should be used for the points? Ignored when `SelectEffect="Transparent"` and the transmitting plot has a non-`NULL` selection of points. Defaults to 1.

`Downsample`: Logical, indicating whether downsampling of overlapping points should be performed. Defaults to `FALSE`.

`SampleRes`: Numeric, specifying the downsampling resolution, i.e., the granularity at which points are considered to overlap. Higher values result in a more stringent definition of overlaps, and thus less downsampling. Defaults to 200.

Text-related parameters are:

`FontSize`: Numeric, size of the font. Defaults to 1.

`LegendPosition`: String specifying the legend position. Defaults to `"Bottom"` but can also be `"Right"`.

`ZoomData`: A list containing numeric vectors of length 4, containing values with names `"xmin"`, `"xmax"`, `"ymin"` and `"ymax"`. These define the zoom window on the x- and y-axes. Each element of the list defaults to `NULL`, i.e., no zooming is performed.

Author(s)

Aaron Lun, Kevin Rue-Albrecht, Charlotte Soneson

See Also

`redDimPlotDefaults`, `featAssayPlotDefaults`, `colDataPlotDefaults`, `rowDataPlotDefaults`, `sampAssayPlotDefaults`

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
redDimPlotDefaults(sce, n=1)
rowDataPlotDefaults(sce, n=1)
```

---

iSEE selection parameters

Selection parameters

Description

Parameters that control the selection of transmitter plots and the effect of selection in a variety of panels.

Selection parameters for plots

`SelectBoxOpen`: Logical, should the selection parameter box be open upon initialization? Defaults to `FALSE`.

`SelectByPlot`: Character, which other plot should be used for point selection in the current plot? Defaults to `"---"`, which means that no plot is used for point selection.

`SelectEffect`: Character, what is the effect of receiving point selection information? Can be `"Restrict"`, where only the selected points are shown; `"Color"`, where the selected points have a different color; or `"Transparent"`, where all points other than those selected are made transparent. Defaults to `"Transparent"`.

`SelectColor`: Character, what color should be used for the selected points when `SelectEffect="Color"`? Defaults to `"red"`.

`SelectAlpha`: Numeric, what level of transparency should be used for the unselected points when `SelectEffect="Transparent"`? This should lie in [0, 1], where 0 is fully transparent and 1 is fully opaque. Defaults to 0.1.

For row-based plots, each point represents a feature, while for column-based plots and heatmaps, each point represents a sample.

Selection information is recorded in the form of brush or lasso objects, which can also be specified:

`BrushData`: A list of shiny brush objects, where each brush object is itself a list - see `brushedPoints` for more details. The list should be of length equal to the number of plots.

`LassoData`: A list of lasso objects, where each lasso object is itself a list - see `lassoPoints` for more details. The list should be of length equal to the number of plots.

Users should not construct these objects by hand, but instead copy-and-paste the reported code generated by `iSEE`'s code tracker.

It is also possible to store multiple alternative selections for point-based plots through the `MultiSelectHistory` field. This should contain a list of lists of brush or lasso objects containing saved alternative selections for each plot. The outer list should be of length equal to the number of plots. Again, users are advised to copy-and-paste reported code rather than writing these objects by hand.

If the transmitting plot has multiple selections, the current panel can choose between them using:

`SelectMultiType`: Character, containing `"Active"`, which uses the active selection on the transmitting panel; `"Union"`, which uses the union of the active selection and all saved selections for the transmitting panel; or `"Saved"`, which uses a single saved selection from the transmitting panel. Defaults to `"Active"`.

`SelectMultiSaved`: Integer, specifying the saved selection to use from the transmitting panel when `SelectMultiType` is set to `"Saved"`. Defaults to 0, i.e., no saved selection is used.

Inter-plot transmission rules

Point selection cannot occur between row-based and column-based plots. This is because each point in a row-based plot is a feature, while each point represents a sample in the other plots. Thus, point selection can only occur between plots of the same point type. The only exception to this rule is the custom panels, which can receive transmissions from both row- and column-based plots – see `customDataPlotDefaults`. This is because they can generate arbitrary plots and are not truly row- or column-based.

Selection parameters for tables

For row or column statistics tables, the following options apply:

`SelectBoxOpen`: Logical, should the point selection parameter box be open upon initialization? Defaults to `FALSE`.

`SelectByPlot`: Character, which other plot should be used to select features in the current table? Defaults to `"---"`, which means that no plot is used for point selection.

`SelectMultiType`: Character, containing `"Active"`, which uses the active selection on the transmitting panel; `"Union"`, which uses the union of the active selection and all saved selections for the transmitting panel; or `"Saved"`, which uses a single saved selection from the transmitting panel.

`SelectMultiSaved`: Integer, specifying the saved selection to use from the transmitting panel when `SelectMultiType` is set to `"Saved"`.

Only row-based plots (i.e., row data and sample assay plots) can be used for selecting points to supply to row statistics tables, for the same reasons described above. Similarly, only column-based plots can be used to select plots to transmit to column statistics tables.

Author(s)

Aaron Lun, Kevin Rue-Albrecht

See Also

redDimPlotDefaults, featAssayPlotDefaults, colDataPlotDefaults, rowDataPlotDefaults, sampAssayPlotDefaults, heatMapPlotDefaults, rowStatTableDefaults

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
redDimPlotDefaults(sce, n=1)
rowStatTableDefaults(sce, n=1)
```

---

iSEE-pkg *iSEE: interactive SummarizedExperiment/SingleCellExperiment Explorer*

Description

iSEE is a Bioconductor package that provides an interactive Shiny-based graphical user interface for exploring data stored in SummarizedExperiment objects, including row- and column-level metadata. Particular attention is given to single-cell data in a SingleCellExperiment object with visualization of dimensionality reduction results, e.g., from principal components analysis (PCA) or t-distributed stochastic neighbour embedding (t-SNE)

Author(s)

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

Charlotte Soneson <charlotte.soneson@uzh.ch>

Federico Marini <marinif@uni-mainz.de>

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

---

jitterSquarePoints *Jitter points for categorical variables*

Description

Add quasi-random jitter on the x-axis for violin plots when the x-axis variable is categorical. Add random jitter within a rectangular area for square plots when both x- and y-axis variables are categorical.

Usage

```
jitterSquarePoints(X, Y, grouping = NULL)

jitterViolinPoints(X, Y, grouping = NULL, ...)
```

Arguments

X	A factor corresponding to a categorical variable.
Y	A numeric vector of the same length as X for jitterViolinPoints, or a factor of the same length as X for jitterSquarePoints.
grouping	A named list of factors of the same length as X, specifying how elements should be grouped.
...	Further arguments to be passed to offsetX.

Details

The jitterViolinPoints function calls offsetX to obtain quasi-random jittered x-axis values. This reflects the area occupied by a violin plot, though some tuning of arguments in ... may be required to get an exact match.

The jitterSquarePoints function will uniformly jitter points on both the x- and y-axes. The jitter area is a square with area proportional to the frequency of the paired levels in X and Y. If either factor only has one level, the jitter area becomes a rectangle that can be interpreted as a bar plot.

If grouping is specified, the values corresponding to each point defines a single combination of levels. Both functions will then perform jittering separately within each unique combination of levels. This is useful for obtaining appropriate jittering when points are split by group, e.g., during faceting.

If grouping!=NULL for jitterSquarePoints the statistics in the returned summary data.frame will be stratified by unique combinations of levels. To avoid clashes with existing fields, the names in grouping should not be "X", "Y", "Freq", "XWidth" or "YWidth".

Value

For jitterViolinPoints, a numeric vector is returned containing the jittered x-axis coordinates for all points.

For jitterSquarePoints, a list is returned with numeric vectors X and Y, containing jittered coordinates on the x- and y-axes respectively for all points; and summary, a data.frame of frequencies and side lengths for each unique pairing of X/Y levels.

Author(s)

Aaron Lun

Examples

```
X <- factor(sample(LETTERS[1:4], 100, replace=TRUE))
Y <- rnorm(100)
(out1 <- jitterViolinPoints(X=X, Y=Y))

Y2 <- factor(sample(letters[1:3], 100, replace=TRUE))
(out2 <- jitterSquarePoints(X=X, Y=Y2))

grouped <- sample(5, 100, replace=TRUE)
(out3 <- jitterViolinPoints(X=X, Y=Y, grouping=list(FacetRow=grouped)))
(out4 <- jitterSquarePoints(X=X, Y=Y2, grouping=list(FacetRow=grouped)))
```

`lassoPoints` *Find rows of data within a closed lasso*

Description

Identify the rows of a data.frame lying within a closed lasso polygon, analogous to `brushedPoints`.

Usage

```
lassoPoints(df, lasso)
```

Arguments

`df`	A data.frame from which to select rows.
`lasso`	A list containing data from a lasso.

Details

This function uses `in.out` from the **mgcv** package to identify points within a polygon. This involves a boundary crossing algorithm that may not be robust in the presence of complex polygons with intersecting edges.

Value

A subset of rows from `df` with coordinates lying within `lasso`.

Author(s)

Aaron Lun

See Also

`brushedPoints`

Examples

```
lasso <- list(coord=rbind(c(0, 0), c(0.5, 0), c(0, 0.5), c(0, 0)),
    closed=TRUE, mapping=list(x="X", y="Y"))
values <- data.frame(X=runif(100), Y=runif(100),
    row.names=sprintf("VALUE_%i", seq_len(100)))
lassoPoints(values, lasso)

# With faceting information:
lasso <- list(coord=rbind(c(0, 0), c(0.5, 0), c(0, 0.5), c(0, 0)),
    panelvar1="A", panelvar2="B", closed=TRUE,
    mapping=list(x="X", y="Y",
    panelvar1="FacetRow", panelvar2="FacetColumn"))
values <- data.frame(X=runif(100), Y=runif(100),
    FacetRow=sample(LETTERS[1:2], 100, replace=TRUE),
    FacetColumn=sample(LETTERS[1:4], 100, replace=TRUE),
    row.names=sprintf("VALUE_%i", seq_len(100)))
lassoPoints(values, lasso)
```

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

Description

App pre-configured to link multiple feature assay plots

Usage

```
modeGating(se, features, featAssayMax = max(2, nrow(features)), ...,
  plot_width = 4)
```

Arguments

se	An object that coercible to SingleCellExperiment
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.
featAssayMax	Maximal number of feature assay plots in the app.
...	Additional arguments passed to iSEE.
plot_width	The grid width of linked plots (numeric vector of length either 1 or equal to nrow(features)

Value

A Shiny App preconfigured with multiple chain-linked feature expression plots is launched for interactive data exploration of the SingleCellExperiment / SummarizedExperiment object

Examples

```
library(scRNAseq)
data(allen)
class(allen)

# Example data ----

library(scater)
sce <- as(allen, "SingleCellExperiment")
counts(sce) <- assay(sce, "tophat_counts")
sce <- normalize(sce)


# Select top variable genes ----

plot_count <- 6
rv <- rowVars(logcounts(sce))
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, featAssayMax=6)
if (interactive()) {
  shiny::runApp(app, port=1234)
}
```

`panelTypes` *Available panel types*

Description

`panelTypes` is the character vector of available panel types. Names indicate the encoded panel name.

`panelCodes` is the complementary of `panelTypes`. Values are the encoded names of each panel type, and names indicate their full name.

Usage

```
panelTypes

panelCodes
```

Format

Named character vectors.

Details

Decoded names are used to define the panels visible at initialization. Refer to the documentation of the `iSEE` function for more information.

Encoded names are used to name the individual HTML elements in the UI. Those names may be used to anchor tour steps at HTML elements using the `rintrojs` package.

Author(s)

Kevin Rue-Albrecht

See Also

`iSEE`

Examples

```
panelTypes
panelCodes
```

prepareSpeechRecognition *Prepare speech recognition*

Description

Prepare speech recognition

Usage

```
prepareSpeechRecognition(use = FALSE)
```

Arguments

use	Whether speech recognition should be enabled.

Value

A list of HTML content to include in the user interface.

Author(s)

Kevin Rue-Albrecht

redDimPlotDefaults *Reduced dimension plot defaults*

Description

Create default settings for reduced dimension plot panels in the iSEE interface.

Usage

```
redDimPlotDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of reduced dimension plots that can be added to the interface.

Details

Parameters available to reduced dimension plots are:

`Type`: Integer, which entry of `reducedDims(se)` should be shown? Defaults to 1, i.e., the first entry. Alternatively, a string can be supplied containing the name of the reduced dimension field, if `reducedDims(se)` has names.

`XAxis`: Integer, which component should be shown on the x-axis? Defaults to 1.

`YAxis`: Integer, which component should be shown on the y-axis? Defaults to 2.

All column-based parameters described in `?"iSEE point parameters"` are applicable. All plot-based parameters described in `?"iSEE selection parameters"` are applicable.

Value

A DataFrame containing default settings for parameters of each of `number` reduced dimension panels.

Author(s)

Aaron Lun

See Also

?"iSEE point parameters", ?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
redDimPlotDefaults(sce, n=1)
```

---

rowDataPlotDefaults *Row data plot defaults*

Description

Create default settings for row data plot panels in the iSEE interface.

Usage

```
rowDataPlotDefaults(se, number)
```

Arguments

`se`	A SummarizedExperiment object.
`number`	An integer scalar, specifying the maximum number of row data plots that can be added to the interface.

Details

Parameters available to row data plots are:

`YAxis`: Character, which column of `rowData(se)` should be shown on the y-axis? Defaults to the first entry of `colData(se)`.

`XAxis`: Character, what variable should be shown on the x-axis? Defaults to `"None"`, but can also be `"Row data"` or `"Feature name"`.

`XAxisRowData`: Character, which column of `rowData(se)` should be shown on the x-axis if `XAxis="Row data"`? Defaults to the first entry of `rowData(se)`.

All row-based parameters described in ?"iSEE point parameters" are applicable. All plot-based parameters described in ?"iSEE selection parameters" are applicable.

Value

A DataFrame containing default settings for parameters of each of `number` row data panels.

Author(s)

Aaron Lun

See Also

?"iSEE point parameters", ?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
rowDataPlotDefaults(sce, n=1)
```

---

rowStatTableDefaults *Row statistics table defaults*

Description

Create default settings for row statistics table panels in the iSEE interface.

Usage

```
rowStatTableDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of row statistics tables that can be added to the interface.

Details

Parameters available to row statistics tables are:

Selected: Integer, containing the index of the row to be initially selected. Defaults to the first row, i.e., 1. Alternatively, a string can be supplied containing the row name.

Search: Character, containing the initial value of the search field. Defaults to an empty string.

SearchColumns: A list containing character vectors of length equal to the number of columns in rowData(se), specifying the initial value of the search field for each column. All entries default to an empty string.

All table-based parameters described in ?"iSEE selection parameters" are applicable.

Value

A DataFrame containing default settings for parameters of each of number row statistics table panels.

Author(s)

Aaron Lun

See Also

?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
rowStatTableDefaults(sce, n=1)
```

---

sampAssayPlotDefaults *Sample assay plot defaults*

Description

Create default settings for sample assay plot panels in the iSEE interface.

Usage

```
sampAssayPlotDefaults(se, number)
```

Arguments

se	A SummarizedExperiment object.
number	An integer scalar, specifying the maximum number of sample assay plots that can be added to the interface.

Details

Parameters available to sample assay plots are:

YAxisSampName: Integer, which column of se should be shown on the y-axis? Defaults to 1, i.e., the first column. Alternatively, a character field can be supplied containing the name of the column.

YAxisColTable: Character, what column statistics table should be used to choose a sample to display on the y-axis? Any setting will override YAxisSampName with the selected row in the chosen table upon initialization of the app. Defaults to "---", which means that no table will be used.

Assay: Integer, which assay should be used to supply the expression values shown on the y-axis? Defaults to 1, i.e., the first assay in se. Alternatively, a string can also be supplied containing the name of the assay, if assays(se) has names.

XAxis: Character, what variable should be shown on the x-axis? Defaults to "None", but can also be "Row data" or "Sample name".

XAxisRowData: Character, which column of rowData(se) should be shown on the x-axis if XAxis="Row data"? Defaults to the first entry of rowData(se).

XAxisSampName: Integer, which column of se should be shown on the x-axis? Defaults to 2 if se contains multiple columns, otherwise it is set to 1. Alternatively, a character field can be supplied containing the name of the column.

XAxisColTable: Character, what column statistics table should be used to choose a sample to display on the x-axis, if XAxis="Sample name"? Any setting will override XAxisSampName with the selected row in the chosen table upon initialization of the app. Defaults to "---", which means that no table will be used.

All row-based parameters described in ?"iSEE point parameters" are applicable. All plot-based parameters described in ?"iSEE selection parameters" are applicable.

Value

A DataFrame containing default settings for parameters of each of number sample assay panels.

Author(s)

Charlotte Soneson

See Also

?"iSEE point parameters", ?"iSEE selection parameters"

Examples

```
example(SingleCellExperiment, echo=FALSE) # mock up 'sce'.
sampAssayPlotDefaults(sce, n=1)
```

---

subsetPointsByGrid *Subset points for faster plotting*

Description

Subset points using a grid-based system, to avoid unnecessary rendering when plotting.

Usage

```
subsetPointsByGrid(X, Y, resolution = 200)
```

Arguments

X	A numeric vector of x-coordinates for all points.
Y	A numeric vector of y-coordinates for all points, of the same length as X.
resolution	A positive integer specifying the number of bins on each axis of the grid.

Details

This function will define a grid of the specified resolution across the plot. Each point is allocated to a grid location (i.e., pair of bins on the x- and y-axes). If multiple points are allocated to a given location, only the last/right-most point is retained. This mimics the fact that plotting will overwrite earlier points with later points. In this manner, we can avoid unnecessary rendering of earlier points that would not show up anyway.

Note that the resolution should be a positive integer less than the square root of the maximum integer size, and will be coerced into the valid range if necessary. This enables fast calculation of the grid locations for all points.

For plots where X and Y are originally categorical, use the jittered versions as input to this function.

Value

A logical vector indicating which points should be retained.

Author(s)

Aaron Lun

Examples

```
X <- rnorm(100000)
Y <- X + rnorm(100000)

summary(subsetPointsByGrid(X, Y, resolution=100))

summary(subsetPointsByGrid(X, Y, resolution=200))

summary(subsetPointsByGrid(X, Y, resolution=1000))
```

---

synchronizeAssays *Synchronize assay colormaps to match those in a SummarizedExperiment*

Description

This function returns an updated ExperimentColorMap in which colormaps in the `assays` slot are ordered to match the position of their corresponding assay in the SingleCellExperiment object. Assays in the SingleCellExperiment that do not have a match in the ExperimentColorMap are assigned the appropriate default colormap.

Usage

```
synchronizeAssays(ecm, se)
```

Arguments

`ecm`	An ExperimentColorMap.
`se`	A SingleCellExperiment.

Details

It is highly recommended to name *all* assays in both ExperimentColorMap and SummarizedExperiment prior to calling this function, as this will facilitate the identification of matching assays between the two objects. In most cases, unnamed colormaps will be dropped from the new ExperimentColorMap object.

The function supports three main situations:

- If *all* assays in the SingleCellExperiment are named, this function will populate the `assays` slot of the new ExperimentColorMap with the name-matched colormap from the input ExperimentColorMap, if available. Assays in the SingleCellExperiment that do not have a colormap defined in the ExperimentColorMap are assigned the appropriate default colormap.
- If *all* assays in the SingleCellExperiment are unnamed, this function requires that the ExperimentColorMap supplies a number of assay colormaps *identical* to the number of assays in the SingleCellExperiment object. In that case, the ExperimentColorMap object will be returned *as is*.

- If only a subset of assays in the SingleCellExperiment are named, this function will ignore unnamed colormaps in the ExperimentColorMap; It will populate the `assays` slot of the new ExperimentColorMap with the name-matched colormap from the input ExperimentColorMap, if available. Assays in the SingleCellExperiment that are unnamed, or that do not have a colormap defined in the ExperimentColorMap are assigned the appropriate default colormap.

Value

An ExperimentColorMap with colormaps in the `assay` slot synchronized to match the position of the corresponding assay in the SingleCellExperiment.

Author(s)

Kevin Rue-Albrecht

Examples

```
# Example ExperimentColorMap ----

count_colors <- function(n){
  c("black","brown","red","orange","yellow")
}
fpkm_colors <- viridis::inferno

ecm <- ExperimentColorMap(
    assays = list(
        counts = count_colors,
        tophat_counts = count_colors,
        cufflinks_fpkm = fpkm_colors,
        rsem_counts = count_colors,
        orphan = count_colors,
        orphan2 = count_colors,
        count_colors,
        fpkm_colors
    )
)

# Example SingleCellExperiment ----

library(scRNAseq)
data(allen)
library(scater)
sce <- as(allen, "SingleCellExperiment")
counts(sce) <- assay(sce, "tophat_counts")
sce <- normalize(sce)
sce <- runPCA(sce)
sce <- runTSNE(sce)

# Example ----

ecm_sync <- synchronizeAssays(ecm, sce)
```

Index