Package 'dittoSeq'

October 17, 2020

Type Package

Title User Friendly Single-Cell and Bulk RNA Sequencing Visualization

Version 1.0.2

Author Daniel Bunis

Maintainer Daniel Bunis <daniel.bunis@ucsf.edu>

Description A universal, user friendly, single-cell and bulk RNA sequencing visualization toolkit that allows highly customizable creation of color blindness friendly, publication-quality figures. dittoSeq accepts both SingleCellExperiment (SCE) and Seurat objects, as well as the import and usage, via conversion to an SCE, of SummarizedExperiment or DGEList bulk data. Visualizations include dimensionality reduction plots, heatmaps, scatterplots, percent composition or expression across groups, and more. Customizations range from size and title adjustments to automatic generation of annotations for heatmaps, overlay of trajectory analysis onto any dimensionality reduciton plot, hidden data overlay upon cursor hovering via ggplotly conversion, and many more. All with simple, discrete inputs. Color blindness friendliness is powered by legend adjustments (enlarged keys), and by allowing the use of shapes or letter-overlay in addition to the carefully selected dittoColors().

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Depends ggplot2

Imports methods, colorspace (>= 1.4), gridExtra, cowplot, reshape2, pheatmap, grDevices, ggrepel, ggridges, stats, utils, SummarizedExperiment, SingleCellExperiment, edgeR, S4Vectors

Suggests plotly, testthat, Seurat (>= 2.2), DESeq2, knitr, BiocStyle, scRNAseq

biocViews Software, Visualization, RNASeq, SingleCell, GeneExpression, Transcriptomics, DataImport

VignetteBuilder knitr

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

git_branch RELEASE_3_11

git_last_commit 28a884c

git_last_commit_date 2020-06-04
Date/Publication 2020-10-16

R topics documented:

addDimReduction . . . 2
addPrcomp . . . 4
Darken . . . 5
demux.calls.summary . . . 6
demux.SNP.summary . . . 7
demuxlet.example . . . 9
dittoBarPlot . . . 9
dittoColors . . . 13
dittoDimPlot . . . 14
dittoHeatmap . . . 21
dittoPlot . . . 25
dittoPlotVarsAcrossGroups . . . 31
dittoScatterPlot . . . 37
dittoSeq . . . 42
gene . . . 43
getGenes . . . 44
getMetas . . . 45
getReductions . . . 46
importDemux . . . 47
importDittoBulk . . . 50
isBulk . . . 53
isGene . . . 54
isMeta . . . 55
Lighten . . . 56
meta . . . 57
metaLevels . . . 58
multi_dittoDimPlot . . . 59
multi_dittoDimPlotVaryCells . . . 60
multi_dittoPlot . . . 63
setBulk . . . 65
Simulate . . . 65

Index **67**

`addDimReduction` *Add any dimensionality reduction space to a SingleCellExperiment object containing bulk or single-cell data*

Description

Add any dimensionality reduction space to a SingleCellExperiment object containing bulk or single-cell data

Usage

```
addDimReduction(object, embeddings, name, key = .gen_key(name))
```

Arguments

object	the bulk or single-cell SingleCellExperiment object to add the dimensionality reduction to. (dittoSeq utilizes the SingleCellExperiment object even for bulk data because it provides a convenient slots for all data that dittoSeq requires)
embeddings	a numeric matrix or matrix-like object, with number of rows equal to ncol(object), containing the coordinates of all cells / samples within the dimensionality reduction space.
name	String name for the reduction slot. Example: "pca". This will become the name of the slot, and what should be provided to the reduction.use input when making a dittoDimPlot. When the name given is the same as that of a slot that already exists inside the object, the previous slot is replaced with the newly provided data.
key	String, like "PC", which sets the default axes-label prefix when this reduction is used for making a dittoDimPlot. If nothing is provided, a key will be automatically generated.

Value

Outputs a SingleCellExperiment object with an added or replaced dimensionality reduction slot.

Author(s)

Daniel Bunis

See Also

addPrcomp for a prcomp specific PCA import wrapper

importDittoBulk for initial import of bulk RNAseq data into dittoSeq as a SingleCellExperiment.

dittoDimPlot for visualizing how samples group within added dimensionality reduction spaces

Examples

```
example("importDittoBulk", echo = FALSE)

# Calculate PCA
#   NOTE: This is typically not done with all genes in the dataset.
#   The inclusion of this example code is not an endorsement of a particular
#   method of PCA. Consult yourself, a bioinformatician, or literature for
#   tips on proper techniques.
embeds <- prcomp(t(logcounts(myRNA)), center = TRUE, scale = TRUE)$x

myRNA <- addDimReduction(
    object = myRNA,
    embeddings = embeds,
    name = "pca",
    key = "PC")

# Visualize conditions metadata on a PCA plot
dittoDimPlot(myRNA, "conditions", reduction.use = "pca", size = 3)
```

addPrcomp *Add a prcomp pca calculation to a SingleCellExperiment object containing bulk or single-cell data*

Description

Add a prcomp pca calculation to a SingleCellExperiment object containing bulk or single-cell data

Usage

```
addPrcomp(object, prcomp, name = "pca", key = "PC")
```

Arguments

object	the SingleCellExperiment object.
prcomp	a prcomp output which will be added to the object
name	String name for the reduction slot. Normally, this will be "pca", but you can hold any number of PCA calculations so long as a unique name is given to each. This will become the name of the slot and what should be provided to the reduction.use input when making a dittoDimPlot. When the name given is the same as that of a slot that already exists inside the object, the previous slot is replaced with the newly provided data.
key	String, like "PC", which sets the default axes-label prefix when this reduction is used for making a dittoDimPlot

Value

Outputs an SingleCellExperiment object with an added or replaced pca reduction slot.

Author(s)

Daniel Bunis

See Also

addDimReduction for adding other types of dimensionality reductions

importDittoBulk for initial import of bulk RNAseq data into dittoSeq as a SingleCellExperiment.

dittoDimPlot for visualizing how samples group within added dimensionality reduction spaces

Examples

```
example("importDittoBulk", echo = FALSE)

# Calculate PCA with prcomp
#   NOTE: This is typically not done with all genes in a dataset.
#   The inclusion of this example code is not an endorsement of a particular
#   method of PCA. Consult yourself, a bioinformatician, or literature for
#   tips on proper techniques.
calc <- prcomp(t(logcounts(myRNA)), center = TRUE, scale = TRUE)
```

```
myRNA <- addPrcomp(
    object = myRNA,
    prcomp = calc)

# Now we can visualize conditions metadata on a PCA plot
dittoDimPlot(myRNA, "conditions", reduction.use = "pca", size = 3)
```

Darken *Darkens input colors by a set amount*

Description

A wrapper for the darken function of the colorspace package.

Usage

```
Darken(colors, percent.change = 0.25, relative = TRUE)
```

Arguments

colors	the color(s) input. Can be a list of colors, for example, /codedittoColors().
percent.change	# between 0 and 1. the percentage to darken by. Defaults to 0.25 if not given.
relative	TRUE/FALSE. Whether the percentage should be a relative change versus an absolute one. Default = TRUE.

Value

Return a darkened version of the color in hexadecimal color form (="#RRGGBB" in base 16)

Author(s)

Daniel Bunis

Examples

```
Darken("blue") #"blue" = "#0000FF"
#Output: "#0000BF"
Darken(dittoColors()[1:8]) #Works for multiple color inputs as well.
```

`demux.calls.summary` *Plots the number of annotations per sample, per lane*

Description

Plots the number of annotations per sample, per lane

Usage

```
demux.calls.summary(
  object,
  singlets.only = FALSE,
  main = "Sample Annotations by Lane",
  sub = NULL,
  ylab = "Annotations",
  xlab = "Sample",
  color = dittoColors()[2],
  theme = NULL,
  rotate.labels = TRUE,
  data.out = FALSE
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object
`singlets.only`	Whether to only show data for cells called as singlets by demuxlet. Default is TRUE. Note: if doublets are included, only one of their sample calls will be used.
`main`	plot title. Default = "Sample Annotations by Lane"
`sub`	plot subtitle
`ylab`	y axis label, default is "Annotations"
`xlab`	x axis label, default is "Sample"
`color`	bars color. Default is the dittoColors skyBlue.
`theme`	A complete ggplot theme. Default is a slightly modified theme_bw().
`rotate.labels`	whether sample names / x-axis labels should be rotated or not. Default is TRUE.
`data.out`	Logical, whether underlying data for the plot should be output instead of the plot itself.

Value

A faceted ggplot summarizing how many cells in each lane were anotated to each sample. Assumes that the Sample calls of each cell, and which lane each cell belonged to, are stored in 'Sample' and 'Lane' metadata slots, respectively, as would be the case if demuxlet information was imported with `importDemux`.

Alternatively, value will be a data.frame containing the underlying data if `data.out = TRUE` is provided.

Author(s)

Daniel Bunis

See Also

`demux.SNP.summary` for plotting the number of SNPs measured per cell. This is the other Demuxlet-associated QC visualization included with dittoSeq.

`importDemux`, for how to import relevant demuxlet information as metadata.

Kang et al. Nature Biotechnology, 2018 https://www.nature.com/articles/nbt.4042 for more information about the demuxlet cell-sample deconvolution method.

Examples

```
example(importDemux, echo = FALSE)

demux.calls.summary(myRNA)

# Exclude doublets by setting 'singlets only = TRUE'
demux.calls.summary(myRNA,
    singlets.only = TRUE)

# To return the underlying data.frame
demux.calls.summary(myRNA, data.out = TRUE)
```

---

demux.SNP.summary *Plots the number of SNPs sequenced per droplet*

Description

Plots the number of SNPs sequenced per droplet

Usage

```
demux.SNP.summary(
  object,
  group.by = "Lane",
  color.by = group.by,
  plots = c("jitter", "boxplot"),
  boxplot.color = "grey30",
  add.line = 50,
  min = 0,
  ...
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object
`group.by`	String "name" of a metadata to use for grouping values. Default is "Lane".
`color.by`	String "name" of a metadata to use for coloring. Default is whatever was provided to `group.by`.

plots	String vector which sets the types of plots to include: possibilities = "jitter", "boxplot", "vlnplot", "ridgeplot". NOTE: The order matters, so use c("back","middle","front") when inputing multiple to put them in the order you want.
boxplot.color	The color of the lines of the boxplot.
add.line	numeric value(s) where a dashed horizontal line should go. Default = 50, a high confidence minimum number of SNPs per cell for highly accurate demuxlet sample deconvolution.
min	numeric value which sets the minimum value shown on the y-axis.
...	extra arguments passed to dittoPlot

Details

This function is a wrapper that essentially runs dittoPlot("demux.N.SNP") with a few modified defaults. The altered defaults:

- Data is grouped and colored by the "Lane" metadata (unless group.by or color.by are adjusted otherwise).
- Data is displayed as boxplots with gray lines on top of dots for individual cells (unless plots or boxplot.color are adjusted otherwise).
- The plot is set to have minimum y axis value of zero (unless min is adjusted otherwise).
- A dashed line is added at the value 50, a very conservative minimum number of SNPs for high confidence sample calls (unless add.line is adjusted otherwise).

Value

A ggplot, made with dittoPlot showing a summary of how many SNPs were available to Demuxlet for each cell of a dataset.

Alternatively, a plotly object if data.hover = TRUE is provided.

Alternatively, list containing a ggplot and the underlying data as a dataframe if data.out = TRUE is provided.

Author(s)

Daniel Bunis

See Also

demux.calls.summary for plotting the number of sample annotations assigned within each lane. This is the other Demuxlet-associated QC visualization included with dittoSeq.

dittoPlot, as demux.SNP.summary is essentially just a dittoPlot wrapper.

importDemux, for how to import relevant demuxlet information as metadata.

Kang et al. Nature Biotechnology, 2018 https://www.nature.com/articles/nbt.4042 for more information about the demuxlet cell-sample deconvolution method.

Examples

```
example(importDemux, echo = FALSE)
demux.SNP.summary(myRNA)

#Function wraps dittoPlot. See dittoPlot docs for more examples
```

`demuxlet.example` *demuxlet.example*

Description

A dataframe containing mock demuxlet information for the 80-cell Seurat::pbmc_small dataset

Usage

```
demuxlet.example
```

Format

An object of class `data.frame` with 80 rows and 7 columns.

Details

This data was created based on the structure of real demuxlet.best output files. Barcodes from the `pbmc_small` dataset were used as the BARCODES column. Cells were then assigned randomly as either SNG (singlets), DBL (doublets), or AMB (ambiguous). Cells were then randomly assign to sample1-10 (or multiple samples for doublets), and this information was combined using the `paste` function into the typical structure of a demuxlet CALL column. Random sampling of remaining data from a separate, actual, demuxlet daatset was used for remaining columns.

Value

A dataframe

Note

This is a slightly simplified example. Real demuxlet.best data has additional columns.

Author(s)

Daniel Bunis

`dittoBarPlot` *Outputs a stacked bar plot to show the percent composition of samples, groups, clusters, or other groupings*

Description

Outputs a stacked bar plot to show the percent composition of samples, groups, clusters, or other groupings

Usage

```
dittoBarPlot(
  object,
  var,
  group.by,
  scale = c("percent", "count"),
  cells.use = NULL,
  data.out = FALSE,
  do.hover = FALSE,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  y.breaks = NA,
  min = 0,
  max = NULL,
  var.labels.rename = NULL,
  var.labels.reorder = NULL,
  x.labels = NULL,
  x.labels.rotate = TRUE,
  x.reorder = NULL,
  theme = theme_classic(),
  xlab = group.by,
  ylab = "make",
  main = "make",
  sub = NULL,
  legend.show = TRUE,
  legend.title = NULL
)
```

Arguments

object	A Seurat or SingleCellExperiment object.
var	String name of a metadata that contains discrete data, or a factor or vector containing such data for all cells/samples in the target object.
group.by	String name of a metadata to use for separating the cells/samples into discrete groups.
scale	"count" or "percent". Sets whether data should be shown as raw counts or scaled to 1 and shown as a percentage.
cells.use	String vector of cells'/samples' names which should be included. Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide USE in colnames(object)[USE] OR object@cell.names[USE]. Note: When cells.use is combined with scale = "percent", left out cells are not considered in calculating percentages. Percents will always total to 1.
data.out	Logical. When set to TRUE, changes the output, from the plot alone, to a list containing the plot ("p") and a data.frame ("data") containing the underlying data. Note: plotly output is turned off in the data.out = TRUE setting, but hover.data is still calculated.
do.hover	Logical which sets whether the ggplot output should be converted to a ggplotly object with data about individual bars displayed when you hover your cursor over them.

`color.panel` String vector which sets the colors to draw from. `dittoColors()` by default.

`colors` Integer vector, which sets the indexes / order, of colors from color.panel to actually use. (Provides an alternative to directly modifying `color.panel`.)

`y.breaks` Numeric vector which sets the plot's tick marks / major gridlines. c(break1,break2,break3,etc.)

`min, max` Scalars which control the zoom of the plot. These inputs set the minimum / maximum values of the y-axis. Default = set based on the limits of the data, 0 to 1 for `scale = "percent"`, or 0 to maximum count for 0 to 1 for `scale = "count"`.

`var.labels.rename`
String vector for renaming the distinct identities of `var` values.

`var.labels.reorder`
Integer vector. A sequence of numbers, from 1 to the number of distinct `var` value idententities, for rearranging the order of labels' groupings within the plot.

Method: Make a first plot without this input. Then, treating the top-most grouping as index 1, and the bottom-most as index n. Values of `var.labels.reorder` should be these indices, but in the order that you would like them rearranged to be.

`x.labels` String vector which will replaceme the x-axis groupings' labels. Regardless of `x.reorder`, the first component of `x.labels` sets the name for the left-most x-axis grouping.

`x.labels.rotate`
Logical which sets whether the x-axis grouping labels should be rotated.

`x.reorder` Integer vector. A sequence of numbers, from 1 to the number of groupings, for rearranging the order of x-axis groupings.

Method: Make a first plot without this input. Then, treating the leftmost grouping as index 1, and the rightmost as index n. Values of `x.reorder` should be these indices, but in the order that you would like them rearranged to be.

`theme` A ggplot theme which will be applied before dittoSeq adjustments. Default = `theme_classic()`. See https://ggplot2.tidyverse.org/reference/ggtheme.html for other options and ideas.

`xlab` String which sets the x-axis title. Default is `group.by` so it defaults to the name of the grouping information. Set to `NULL` to remove.

`ylab` String which sets the y-axis title.

`main` String, sets the plot title

`sub` String, sets the plot subtitle

`legend.show` Logical which sets whether the legend should be displayed.

`legend.title` String which adds a title to the legend.

Details

The function creates a dataframe containing counts and percent makeup of `var` identities for each x-axis grouping (determined by the `group.by` input). If a set of cells/samples to use is indicated with the `cells.use` input, only those cells/samples are used for counts and percent makeup calculations. Then, a vertical bar plot is generated (`ggplot2::geom_col()`) showing either percent makeup if `scale = "percent"`, which is the default, or raw counts if `scale = "count"`.

Value

A ggplot plot where discrete data, grouped by sample, condition, cluster, etc. on the x-axis, is shown on the y-axis as either counts or percent-of-total-per-grouping in a stacked barplot.

Alternatively, if `data.out = TRUE`, a list containing the plot ("p") and a dataframe of the underlying data ("data").

Alternatively, if `do.hover = TRUE`, a plotly conversion of the ggplot output in which underlying data can be retrieved upon hovering the cursor over the plot.

Many characteristics of the plot can be adjusted using discrete inputs

- Colors can be adjusted with `color.panel` and/or `colors`.
- y-axis zoom and tick marks can be adjusted using `min`, `max`, and `y.breaks`.
- Titles can be adjusted with `main`, `sub`, `xlab`, `ylab`, and `legend.title` arguments.
- The legend can be removed by setting `legend.show = FALSE`.
- x-axis labels and groupings can be changed / reordered using `x.labels` and `x.reorder`, and rotation of these labels can be turned off with `x.labels.rotate = FALSE`.
- y-axis `var`-group labels and their order can be changed / reordered using `var.labels` and `var.labels.reorder`.

Author(s)

Daniel Bunis

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

dittoBarPlot(myRNA, "clustering", group.by = "groups")
dittoBarPlot(myRNA, "clustering", group.by = "groups",
    scale = "count")

# Reordering the x-axis groupings to have "C" (#3) come first
dittoBarPlot(myRNA, "clustering", group.by = "groups",
    x.reorder = c(3,1,2,4))

### Accessing underlying data:
# as dataframe
dittoBarPlot(myRNA, "clustering", group.by = "groups",
    data.out = TRUE)
# through hovering the cursor over the relevant parts of the plot
if (requireNamespace("plotly", quietly = TRUE)) {
    dittoBarPlot(myRNA, "clustering", group.by = "groups",
        do.hover = TRUE)
    }
```

`dittoColors` *Extracts the dittoSeq default colors*

Description

Creates a string vector of 40 unique colors, in hexadecimal form, repeated 100 times. Or, if `get.names` is set to `TRUE`, outputs the names of the colors which can be helpful as reference when adjusting how colors get used.

These colors are a modification of the protanope and deuteranope friendly colors from Wong, B. Nature Methods, 2011.

Truly, only the first 1-7 are maximally (red-green) color-blindness friendly, but the lightened and darkened versions (plus grey) in slots 8-40 still work releatively well at extending their utility further. Note that past 40, the colors simply repeat in order to most easily allow dittoSeq visualizations to handle situations requiring even more colors.

The colors are:

1-7 = Suggested color panel from Wong, B. Nature Methods, 2011, minus black

- 1- orange = "#E69F00"
- 2- skyBlue = "#56B4E9"
- 3- bluishGreen = "#009E73"
- 4- yellow = "#F0E442"
- 5- blue = "#0072B2"
- 6- vermillion = "#D55E00"
- 7- reddishPurple = "#CC79A7"

8 = gray40

9-16 = 25% darker versions of colors 1-8

17-24 = 25% lighter versions of colors 1-8

25-32 = 40% lighter versions of colors 1-8

33-40 = 40% darker versions of colors 1-8

Usage

```
dittoColors(reps = 100, get.names = FALSE)
```

Arguments

`reps`	Integer which sets how many times the original set of colors should be repeated
`get.names`	Logical, whether only the names of the default dittoSeq color panel should be returned instead

Value

A string vector with length = 24.

Author(s)

Daniel Bunis

Examples

```
dittoColors()

#To retrieve names:
dittoColors(get.names = TRUE)
```

`dittoDimPlot` *Shows data overlayed on a tsne, pca, or similar type of plot*

Description

Shows data overlayed on a tsne, pca, or similar type of plot

Usage

```
dittoDimPlot(
  object,
  var,
  reduction.use = .default_reduction(object),
  size = 1,
  opacity = 1,
  dim.1 = 1,
  dim.2 = 2,
  cells.use = NULL,
  shape.by = NULL,
  split.by = NULL,
  extra.vars = NULL,
  split.nrow = NULL,
  split.ncol = NULL,
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  shape.panel = c(16, 15, 17, 23, 25, 8),
  show.others = TRUE,
  show.axes.numbers = TRUE,
  show.grid.lines = !grepl("umap|tsne", tolower(reduction.use)),
  main = "make",
  sub = NULL,
  xlab = "make",
  ylab = "make",
  theme = theme_bw(),
  legend.show = TRUE,
  legend.size = 5,
  legend.title = "make",
  shape.legend.size = 5,
  shape.legend.title = shape.by,
  do.ellipse = FALSE,
  do.label = FALSE,
  labels.size = 5,
```

```
  labels.highlight = TRUE,
  labels.repel = TRUE,
  rename.var.groups = NULL,
  rename.shape.groups = NULL,
  min.color = "#F0E442",
  max.color = "#0072B2",
  min = NULL,
  max = NULL,
  legend.breaks = waiver(),
  legend.breaks.labels = waiver(),
  do.letter = FALSE,
  do.hover = FALSE,
  hover.data = var,
  hover.assay = .default_assay(object),
  hover.slot = .default_slot(object),
  hover.adjustment = NULL,
  add.trajectory.lineages = NULL,
  add.trajectory.curves = NULL,
  trajectory.cluster.meta,
  trajectory.arrow.size = 0.15,
  data.out = FALSE
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object to work with
`var`	String name of a "gene" or "metadata" (or "ident" for a Seurat `object`) to use for coloring the plots. This is the data that will be displayed for each cell/sample. Discrete or continuous data both work. Alternatively, can be a vector of same length as there are cells/samples in the `object`.
`reduction.use`	String, such as "pca", "tsne", "umap", or "PCA", etc, which is the name of a dimensionality reduction slot within the object, and which sets what dimensionality reduction space within the object to use. Default = the first dimensionality reduction slot inside the object named "umap", "tsne", or "pca", or the first dimensionality reduction slot if none of those exist.
`size`	Number which sets the size of data points. Default = 1.
`opacity`	Number between 0 and 1. Great for when you have MANY overlapping points, this sets how solid the points should be: 1 = not see-through at all. 0 = invisible. Default = 1. (In terms of typical ggplot variables, = alpha)
`dim.1`	The component number to use on the x-axis. Default = 1
`dim.2`	The component number to use on the y-axis. Default = 2
`cells.use`	String vector of cells'/samples' names which should be included. Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide `USE` in `colnames(object)[USE]` OR `object@cell.names[USE]`.
`shape.by`	Variable for setting the shape of cells/samples in the plot. Note: must be discrete. Can be the name of a gene or meta-data. Alternatively, can be "ident" for clusters of a Seurat object. Alternatively, can be a numeric of length equal to the total number of cells/samples in object.

Note: shapes can be harder to see, and to process mentally, than colors. Even as a color blind person myself writing this code, I recommend use of colors for variables with many discrete values.

`split.by` 1 or 2 strings naming discrete metadata to use for splitting the cells/samples into multiple plots with ggplot faceting.

When 2 metadatas are named, c(row,col), the first is used as rows and the second is used for columns of the resulting grid.

When 1 metadata is named, shape control can be achieved with `split.nrow` and `split.ncol`

`extra.vars` String vector providing names of any extra metadata to be stashed in the dataframe supplied to `ggplot(data)`.

Useful for making custom spliting/faceting or other additional alterations *after* dittoSeq plot generation.

`split.nrow, split.ncol`

Integers which set the dimensions of faceting/splitting when a single metadata is given to `split.by`.

`assay, slot` single strings or integer that set which data to use when plotting gene expression. See `gene` for more information.

`adjustment` When plotting gene expression (or antibody, or other forms of counts data), should that data be used directly (default) or should it be adjusted to be

- "z-score": scaled with the scale() function to produce a relative-to-mean z-score representation
- "relative.to.max": divided by the maximum expression value to give percent of max values between [0,1]

`color.panel` String vector which sets the colors to draw from. `dittoColors()` by default, see `dittoColors` for contents.

`colors` Integer vector, the indexes / order, of colors from color.panel to actually use.

Useful for quickly swapping the colors of nearby clusters.

`shape.panel` Vector of integers corresponding to ggplot shapes which sets what shapes to use. When discrete groupings are supplied by `shape.by`, this sets the panel of shapes. When nothing is supplied to `shape.by`, only the first value is used. Default is a set of 6, `c(16,15,17,23,25,8)`, the first being a simple, solid, circle.

Note: Unfortunately, shapes can be hard to see when points are on top of each other & they are more slowly processed by the brain. For these reasons, even as a color blind person myself writing this code, I recommend use of colors for variables with many discrete values.

`show.others` Logical. Whether other cells should be shown in the background in light gray. Default = TRUE.

`show.axes.numbers`

Logical which controls whether the axes values should be displayed.

`show.grid.lines`

Logical which sets whether gridlines of the plot should be shown. They are removed when set to FALSE. Default = TRUE for umap and tsne `reduction.use`, FALSE otherwise.

`main` String, sets the plot title. Default title is automatically generated if not given a specific value. To remove, set to `NULL`.

`sub` String, sets the plot subtitle

`xlab, ylab` Strings which set the labels for the axes. Default labels are generated if you do not give this a specific value. To remove, set to `NULL`.

`theme` A ggplot theme which will be applied before dittoSeq adjustments. Default = `theme_bw()`. See https://ggplot2.tidyverse.org/reference/ggtheme.html for other options and ideas.

`legend.show` Logical. Whether the legend should be displayed. Default = `TRUE`.

`legend.size` Number representing the size at which color legend shapes should be plotted (for discrete variable plotting) in the color legend. Default = 5. *Enlarging the colors legend is incredibly helpful for making colors more distinguishable by color blind individuals.

`legend.title` String which sets the title for the color legend. Default = `NULL` normally, but `var` when a shape legend will also be shown.

`shape.legend.size`
Number representing the size at which shapes should be plotted in the shape legend.

`shape.legend.title`
String which sets the title of the shapes legend. Default is `shape.by`

`do.ellipse` Logical. Whether the groups should be surrounded by median-centered ellipses.

`do.label` Logical. Whether to add text labels near the center (median) of clusters for grouping vars.

`labels.size` Size of the the labels text

`labels.highlight`
Logical. Whether the labels should have a box behind them

`labels.repel` Logical, that sets whether the labels' placements will be adjusted with ggrepel to avoid intersections between labels and plot bounds. TRUE by default.

`rename.var.groups`
String vector which sets new names for the identities of `var` groups.

`rename.shape.groups`
String vector which sets new names for the identities of `shape.by` groups.

`min.color` color for lowest values of `var`/`min`. Default = yellow

`max.color` color for highest values of `var`/`max`. Default = blue

`min` Number which sets the value associated with the minimum color.

`max` Number which sets the value associated with the maximum color.

`legend.breaks` Numeric vector which sets the discrete values to show in the color-scale legend for continuous data.

`legend.breaks.labels`
String vector, with same length as `legend.breaks`, which renames what's displayed next to the tick marks of the color-scale.

`do.letter` Logical which sets whether letters should be added on top of the colored dots. For extended colorblindness compatibility. NOTE: `do.letter` is ignored if `do.hover` = `TRUE` or `shape.by` is provided a metadata because lettering is incompatible with plotly and with changing the dots' to be different shapes.

`do.hover` Logical which controls whether the output will be converted to a plotly object so that data about individual points will be displayed when you hover your cursor over them. `hover.data` argument is used to determine what data to use.

`hover.data` String vector of gene and metadata names, example: `c("meta1","gene1","meta2")` which determines what data to show on hover when `do.hover` is set to `TRUE`.

`hover.assay`, `hover.slot`, `hover.adjustment`
: Similar to the non-hover versions of these inputs, when showing expression data upon hover, these set what data will be shown.

`add.trajectory.lineages`
: List of vectors representing trajectory paths, each from start-cluster to end-cluster, where vector contents are the names of clusters provided in the `trajectory.cluster.meta` input.

  If the `slingshot` package was used for trajectory analysis, you can use `add.trajectory.lineages = SlingshotDataSet(SCE_with_slingshot)$lineages`. In future versions, I might build such retrieval in by default for SCEs.

`add.trajectory.curves`
: List of matrices, each representing coordinates for a trajectory path, from start to end, where matrix columns represent x (`dim.1`) and y (`dim.2`) coordinates of the paths.

  Alternatively, a list of lists(/princurve objects) can be provided. Thus, if the `slingshot` package was used for trajectory analysis, you can provide `add.trajectory.curves = SlingshotDataSet(SCE_with_slingshot)$curves`

`trajectory.cluster.meta`
: String name of metadata containing the clusters that were used for generating trajectories. Required when plotting trajectories using the `add.trajectory.lineages` method. Names of clusters inside the metadata should be the same as the contents of `add.trajectory.lineages` vectors.

`trajectory.arrow.size`
: Number representing the size of trajectory arrows, in inches. Default = 0.15.

`data.out`
: Logical. When set to `TRUE`, changes the output, from the plot alone, to a list containing the plot ("p"), a data.frame containing the underlying data for target cells ("Target_data"), and a data.frame containing the underlying data for non-target cells ("Others_data").

  Note: `do.hover` plotly conversion is turned off in this setting, but hover.data is still calculated.

Details

The function creates a dataframe containing the metadata or expression data associated with the given `var` (or if a vector of data is provided directly, it just uses that), plus X and Y coordinates data determined by the `reduction.use` and `dim.1` (x-axis) and `dim.2` (y-axis) inputs. Any extra data requested with `shape.by`, `split.by` or `extra.var` is added as well. For expression/counts data, `assay`, `slot`, and `adjustment` inputs can be used to change which data is used, and if it should be adjusted in some way.

Next, if a set of cells or samples to use is indicated with the `cells.use` input, then the dataframe is split into `Target_data` and `Others_data` based on subsetting by the target cells/samples.

Finally, a scatter plot is then created using these dataframes where non-target cells will be displayed in gray if `show.others=TRUE`, and target cell data is displayed on top, colored based on the `var`-associated data, and with shapes determined by the `shape.by`-associated data. If `split.by` was used, the plot will be split into a matrix of panels based on the associated groupings.

Value

A ggplot or plotly object where colored dots (or other shapes) are overlayed onto a tSNE, PCA, UMAP, ..., plot of choice.

Alternatively, if `data.out=TRUE`, a list containing three slots is output: the plot (named 'p'), a data.table containing the underlying data for target cells (named 'Target_data'), and a data.table containing the underlying data for non-target cells (named 'Others_data').

Alternatively, if `do.hover` is set to `TRUE`, the plot is coverted from ggplot to plotly & cell/sample information, determined by the `hover.data` input, is retrieved, added to the dataframe, and displayed upon hovering the cursor over the plot.

Many characteristics of the plot can be adjusted using discrete inputs

- `size` and `opacity` can be used to adjust the size and transparency of the data points.
- Color can be adjusted with `color.panel` and/or `colors` for discrete data, or `min`, `max`, `min.color`, and `max.color` for continuous data.
- Shapes can be adjusted with `shape.panel`.
- Color and shape labels can be changed using `rename.var.groups` and `rename.shape.groups`.
- Titles and axes labels can be adjusted with `main`, `sub`, `xlab`, `ylab`, and `legend.title` arguments.
- Legends can also be adjusted in other ways, using variables that all start with `"legend."` for easy tab-completion lookup.

Additional Features

Many other tweaks and features can be added as well. Each is accessible through 'tab' autocompletion starting with `"do."---` or `"add."---`, and if additional inputs are involved in implementing or tweaking these, the associated inputs will start with the `"---."`:

- If `do.label` is set to `TRUE`, labels will be added based on median centers of the discrete `var`-data groupings. The size of the text in the labels can be adjusted using the `labels.size` input. By default labels will repel eachother and the bounds of the plot, and labels will be highlighted with a white background. Either of these can be turned off by setting `labels.repel = FALSE` or `labels.highlight = FALSE`,
- If `do.ellipse` is set to `TRUE`, ellipses will be added to highlight distinct `var`-data groups' positions based on median positions of their cell/sample components.
- If `add.trajectory.lineages` is provided a list of vectors (each vector being cluster names from start-cluster-name to end-cluster-name), and a metadata name pointing to the relevant clustering information is provided to `trajectory.cluster.meta`, then median centers of the clusters will be calculated and arrows will be overlayed to show trajectory inference paths in the current dimmenionality reduction space.
- If `add.trajectory.curves` is provided a list of matrices (each matrix containing x, y coordinates from start to end), paths and arrows will be overlayed to show trajectory inference curves in the current dimmenionality reduction space. Arrow size is controlled with the `trajectory.arrow.size` input.

Author(s)

Daniel Bunis

See Also

`getGenes` and `getMetas` to see what the `var`, `shape.by`, etc. options are.

importDittoBulk for how to create a SingleCellExperiment object from bulk seq data that dittoSeq functions can use & addDimReduction for how to specifically add calculated dimensionality reductions that dittoDimPlot can utilize.

dittoScatterPlot for showing very similar data representations, but where genes or metadata are wanted as the axes.

dittoPlot for an alternative continuous data display method where data is shown on a y- (or x-) axis.

dittoBarPlot for an alternative discrete data display and quantification method.

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

# Display discrete data:
dittoDimPlot(myRNA, "clustering")
# Display continuous data:
dittoDimPlot(myRNA, "gene1")

# To show currently set clustering for seurat objects, you can use "ident".
# To change the dimensional reduction type, use 'reduction.use'.
dittoDimPlot(myRNA, "clustering",
    reduction.use = "pca",
    dim.1 = 3,
    dim.2 = 4)

# Subset to certain cells with cells.use
dittoDimPlot(myRNA, "clustering",
    cells.us = !myRNA$SNP)

# Data can also be split in other ways with 'shape.by' or 'split.by'
dittoDimPlot(myRNA, "gene1",
    shape.by = "clustering",
    split.by = "SNP") # single split.by element
dittoDimPlot(myRNA, "gene1",
    split.by = c("groups","SNP")) # row and col split.by elements

# Modify the look with intuitive inputs
dittoDimPlot(myRNA, "clustering",
    size = 2, opacity = 0.7, show.axes.numbers = FALSE,
    ylab = NULL, xlab = "tSNE",
    main = "Plot Title",
    sub = "subtitle",
    legend.title = "clustering")

# MANY addtional tweaks are possible.
# Also, many extra features are easy to add as well:
dittoDimPlot(myRNA, "clustering",
    do.label = TRUE, do.ellipse = TRUE)
dittoDimPlot(myRNA, "clustering",
    do.label = TRUE, labels.highlight = FALSE, labels.size = 8)
```

```
if (requireNamespace("plotly", quietly = TRUE)) {
    dittoDimPlot(myRNA, "gene1", do.hover = TRUE,
        hover.data = c("gene2", "clustering", "timepoint"))
}
dittoDimPlot(myRNA, "gene1", add.trajectory.lineages = list(c(1,2,4), c(1,3)),
    trajectory.cluster.meta = "clustering",
    sub = "Pseudotime Trajectories")
```

---

dittoHeatmap *Outputs a heatmap of given genes*

Description

Given a set of genes, cells/samples, and metadata names for column annotations, this function will retrieve the expression data for those genes and cells, and the annotation data for those cells. It will then utilize these data to make a heatmap using the pheatmap function of the pheatmap package.

Usage

```
dittoHeatmap(
  object,
  genes = getGenes(object, assay),
  cells.use = NULL,
  annot.by = NULL,
  order.by = .default_order(object, annot.by),
  main = NA,
  cell.names.meta = NULL,
  assay = .default_assay(object),
  slot = .default_slot(object),
  heatmap.colors = colorRampPalette(c("blue", "white", "red"))(50),
  scaled.to.max = FALSE,
  heatmap.colors.max.scaled = colorRampPalette(c("white", "red"))(25),
  annot.colors = c(dittoColors(), dittoColors(1)[seq_len(7)]),
  annotation_col = NULL,
  annotation_colors = NULL,
  data.out = FALSE,
  highlight.genes = NULL,
  show_colnames = isBulk(object),
  show_rownames = TRUE,
  scale = "row",
  cluster_cols = isBulk(object),
  border_color = NA,
  legend_breaks = NA,
  breaks = NA,
  ...
)
```

Arguments

object	A Seurat or SingleCellExperiment object to work with
genes	String vector, c("gene1","gene2","gene3",...) = the list of genes to put in the heatmap. If not provided, defaults to all genes of the object / assay.

`cells.use` String vector of cells'/samples' names which should be included.
Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include.
For the typically easier logical method, provide `USE` in `colnames(object)[USE]` OR `object@cell.names[USE]`.

`annot.by` String name of any metadata slots containing how the cells/samples should be annotated.

`order.by` Single string or numeric vector which sets the ordering of cells/samples. Can be the name of a gene, or metadata slot. Alternatively, can be a numeric vector of length equal to the total number of cells/samples in object.

`main` String that sets the title for the heatmap.

`cell.names.meta`
quoted "name" of a meta.data slot to use for naming the columns instead of using the raw cell/sample names.

`assay, slot` single strings or integer that set which expression data to use. See `gene` for more information about how defaults for these are filled in when not provided.

`heatmap.colors` the colors to use within the heatmap when (default setting) `scaled.to.max` is set to `FALSE`. Default is a ramp from navy to white to red with 50 slices.

`scaled.to.max` Logical, `FALSE` by default, which sets whether expression shoud be scaled between [0, 1]. This is recommended for single-cell datasets as they are generally enriched in 0s.

`heatmap.colors.max.scaled`
the colors to use within the heatmap when `scaled.to.max` is set to `TRUE`. Default is a ramp from white to red with 25 slices.

`annot.colors` String (color) vector where each color will be assigned to an individual annotation in the generated annotation bars.

`data.out` Logical. When set to `TRUE`, changes the output from the heatmat itself, to a list containing all arguments that would have be passed to `pheatmap` for heatmap generation. (Can be useful for troubleshooting or customization.)

`highlight.genes`
String vector of genes whose names you would like to show. Only these genes will be named in the resulting heatmap.

`show_colnames, show_rownames, scale, annotation_col, annotation_colors`
arguments passed to `pheatmap` that are over-ruled by certain `dittoHeatmap` functionality:

- show_colnames (& labels_col): if `cell.names.meta` is provided, pheatmaps's `labels_col` is utilized to show these names and `show_colnames` parameter is set to `TRUE`.
- show_rownames (& labels_row): if feature names are provided to `highlight.genes`, pheatmap's `labels_row` is utilized to show just these features' names and `show_rownames` parameter is set to `TRUE`.
- scale: when parameter `scaled.to.max` is set to true, pheatmap's `scale` is set to `"none"` and the max scaling is performed prior to the pheatmap call.
- annotation_col: Can be provided as normal by the user and any metadata given to `annot.by` will then be appended.
- annotation_colors: dittoHeatmap fills this complicated-to-produce input in automatically by pulling from the colors given to `annot.colors`, but it is possible to set all or some manually. dittoSeq will just fill any left out annotations. Format is a named (annotation_col & annotation_row colnames) character vector list where individual color values can also be named.

`cluster_cols`, `border_color`, `legend_breaks`, `breaks`, `...`
other arguments passed to `pheatmap` directly.

Details

This function serves as a wrapper for creating heatmaps from bulk or single-cell RNAseq data with `pheatmap`, by essentially automating the data extraction and annotation building steps.

The function will extract the expression matrix for a set of `genes` and/or an optional subset of cells / samples to use via `cells.use`, This matrix is either left as is, default (for scaling within the ultimate call to pheatmap), or if `scaled.to.max = TRUE`, is scaled by dividing each row by its maximum value.

When provided with a set of metadata slot names to use for building annotations (with the `annot.by` input), the relevant metadata is retrieved from the `object` and compiled into a `pheatmap`-ready `annotation_col` input. The input `annot.colors` is used to establish the set of colors that should be used for building a `pheatmap`-ready `annotation_colors` input as well, unless such an input has been provided by the user. See below for further details.

Value

A `pheatmap` object.

Alternatively, if `data.out` was set to `TRUE`, a list containing all arguments that would have be passed to pheatmap to generate such a heatmap.

Many additional characteristics of the plot can be adjusted using discrete inputs

- The cells can be ordered in a set way using the `order.by` input.
  Such ordering happens by default for single-cell RNAseq data when any metadata are provided to `annot.by` as it is often unfeasible to cluster thousands of cells.
- A plot title can be added with `main`.
- **Gene or cell/sample names** can be hidden with `show_rownames` and `show_colnames`, respectively, or...
  - Particular genes can also be selected for labeling using the `highlight.genes` input.
  - Names of all cells/samples can be replaced with the contents of a metadata slot using the `cell.names.meta` input.
- Additional tweaks are possible through use of `pheatmap` inputs which will be directly passed through. Some examples of useful `pheatmap` parameters are:
  - `cluster_cols` and `cluster_rows` for controlling clustering. Note: cluster_cols will always be over-written to be `FALSE` when the input `order.by` is used above.
  - `treeheight_row` and `treeheight_col` for setting how large the trees on the side/top should be drawn.
  - `cutree_col` and `cutree_row` for spliting the heatmap based on kmeans clustering

Customized annotations

In typical operation, dittoHeatmap pulls metadata annotations given to `annot.by` to build a pheatmap-`annotation_col` input, then it uses the colors provided to `annot.colors` to create the pheatmap-`annotation_colors` input which sets the annotation coloring. Specifically...

- colors for the values of **discrete** metadata are pulled from the *start* of the `annot.colors` vector, in the order that they are given to `annot.by`

- colors for the values of **continuous** metadata are pulled from the *end* of the `annot.colors` vector, in the order that they are given to `annot.by`

To customize colors or add additional column or row annotations, users can also provide `annotation_colors`, `annotation_col`, or `annotation_row` pheatmap-inputs directly. General structure is described below, but see `pheatmap` for additional details and examples.

- `annotation_col` = a data.frame with rownames of the barcodes/names of all cells/samples in the dataset & columns representing annotations. Names of columns are used as the annotation titles. *dittoSeq will append any `annot.by` annotations to this dataframe.
- `annotation_row` = a data.frame with rownames of the genes/feature of the dataset & columns representing annotations. Names of columns are used as the annotation titles.
- `annotation_colors` = a named list of string (color) vectors. Vectors must be named by the row or column annotation title that they are associated with. Optionally, individual colors can be named with the values that they should be associated with.

  Partial `annotation_colors` lists (containing vectors for only certain annotations) will have colors for left out annotations filled in automatically. For such filling, `annot.colors` are pulled for column annotations first, then for row annotations.

Author(s)

Daniel Bunis

See Also

`pheatmap`, for how to add additional heatmap tweaks.

`metaLevels` for helping to create manual annotation_colors inputs. This function universally checks the options/levels of a string, factor (filled only by default), or numerical metadata.

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA
scRNA <- setBulk(myRNA, FALSE)

# Pick a set of genes
genes <- getGenes(myRNA)[1:30]

# Make a heatmap with cells/samples annotated by their clusters
dittoHeatmap(myRNA, genes,
    annot.by = "clustering")

# For single-cell data, you will typically have more cells than can be
# clustered quickly. Thus, cell clustering is turned off by default for
# single-cell data.
dittoHeatmap(scRNA, genes,
    annot.by = "clustering")

# Using the 'order.by' input:
#   ordering by a useful metadata or gene is generally more helpful
```

```
#   For single-cell data, order.by defaults to the first element given to
#     annot.by.
#   For bulk data, order.by must be set separately.
dittoHeatmap(myRNA, genes,
    annot.by = "clustering",
    order.by = "clustering")

# When there are many cells, showing names becomes less useful.
#   Names can be turned off with the show_colnames parameter.
dittoHeatmap(myRNA, genes,
    annot.by = "groups",
    order.by = "groups",
    show_colnames = FALSE)

# Additionally, it is recommended for single-cell data that the parameter
#   scaled.to.max be set to TRUE, or scale be "none" and turned off altogether,
#   because these data are generally enriched for zeros that otherwise get
#   scaled to a negative value.
dittoHeatmap(myRNA, genes, annot.by = "groups",
    order.by = "groups", show_colnames = FALSE,
    scaled.to.max = TRUE)
```

`dittoPlot` *Plots continuous data for cutomizable cells'/samples' groupings on a y-axis*

Description

Plots continuous data for cutomizable cells'/samples' groupings on a y-axis

Usage

```
dittoPlot(
  object,
  var,
  group.by,
  color.by = group.by,
  shape.by = NULL,
  split.by = NULL,
  extra.vars = NULL,
  cells.use = NULL,
  plots = c("jitter", "vlnplot"),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  do.hover = FALSE,
  hover.data = var,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  shape.panel = c(16, 15, 17, 23, 25, 8),
  theme = theme_classic(),
  main = "make",
```

```
  sub = NULL,
  ylab = "make",
  y.breaks = NULL,
  min = NULL,
  max = NULL,
  xlab = group.by,
  x.labels = NULL,
  x.labels.rotate = NA,
  x.reorder = NULL,
  split.nrow = NULL,
  split.ncol = NULL,
  jitter.size = 1,
  jitter.width = 0.2,
  jitter.color = "black",
  jitter.shape.legend.size = NA,
  jitter.shape.legend.show = TRUE,
  boxplot.width = 0.2,
  boxplot.color = "black",
  boxplot.show.outliers = NA,
  boxplot.fill = TRUE,
  vlnplot.lineweight = 1,
  vlnplot.width = 1,
  vlnplot.scaling = "area",
  ridgeplot.lineweight = 1,
  ridgeplot.scale = 1.25,
  add.line = NULL,
  line.linetype = "dashed",
  line.color = "black",
  legend.show = TRUE,
  legend.title = "make",
  data.out = FALSE
)

dittoRidgePlot(..., plots = c("ridgeplot"))

dittoRidgeJitter(..., plots = c("ridgeplot", "jitter"))

dittoBoxPlot(..., plots = c("boxplot", "jitter"))
```

Arguments

`object`	A Seurat or SingleCellExperiment object to work with
`var`	Single string representing the name of a metadata or gene, OR a vector with length equal to the total number of cells/samples in the dataset. This is the data that will be displayed.
`group.by`	String representing the name of a metadata to use for separating the cells/samples into discrete groups.
`color.by`	String representing the name of a metadata to use for setting fills. Great for highlighting subgroups when wanted, but it defaults to `group.by` so this input can be skipped otherwise. Affects boxplot, vlnplot, and ridgeplot fills.
`shape.by`	Single string representing the name of a metadata to use for setting the shapes of the jitter points. When not provided, all cells/samples will be represented with

dots.

`split.by` 1 or 2 strings naming discrete metadata to use for splitting the cells/samples into multiple plots with ggplot faceting.

When 2 metadatas are named, c(row,col), the first is used as rows and the second is used for columns of the resulting grid.

When 1 metadata is named, shape control can be achieved with `split.nrow` and `split.ncol`

`extra.vars` String vector providing names of any extra metadata to be stashed in the dataframe supplied to `ggplot(data)`.

Useful for making custom spliting/faceting or other additional alterations *after* dittoSeq plot generation.

`cells.use` String vector of cells'/samples' names which should be included. Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide `USE` in `colnames(object)[USE]` OR `object@cell.names[USE]`.

`plots` String vector which sets the types of plots to include: possibilities = "jitter", "boxplot", "vlnplot", "ridgeplot". Order matters: c("vlnplot", "boxplot", "jitter") will put a violin plot in the back, boxplot in the middle, and then individual dots in the front. See details section for more info.

`assay, slot` single strings or integer that set which data to use when plotting gene expression / feature data. See `gene` for more information.

`adjustment` When plotting gene expression / feature counts, should that data be used directly (default) or should it be adjusted to be

- "z-score": scaled with the scale() function to produce a relative-to-mean z-score representation
- "relative.to.max": divided by the maximum expression value to give percent of max values between [0,1]

`do.hover` Logical. Default = `FALSE`. If set to `TRUE` (and if there is a "jitter" in `plots`): object will be converted to a ggplotly object so that data about individual cells will be displayed when you hover your cursor over the jitter points,

Note: Currently, hovering is incompatible with RidgePlots as plotly does not support the ggplot geom.

`hover.data` String vector, a list of variable names, c("meta1","gene1","meta2",...) which determines what data to show upon hover when do.hover is set to `TRUE`.

`color.panel` String vector which sets the colors to draw from for plot fills. Default = `dittoColors()`.

`colors` Integer vector, the indexes / order, of colors from color.panel to actually use. (Provides an alternative to directly modifying `color.panel`.)

`shape.panel` Vector of integers corresponding to ggplot shapes which sets what shapes to use. When discrete groupings are supplied by `shape.by`, this sets the panel of shapes which will be used. When nothing is supplied to `shape.by`, only the first value is used. Default is a set of 6, `c(16,15,17,23,25,8)`, the first being a simple, solid, circle.

`theme` A ggplot theme which will be applied before dittoSeq adjustments. Default = `theme_classic()`. See `https://ggplot2.tidyverse.org/reference/ggtheme.html` for other options and ideas.

`main` String, sets the plot title. Default = "make" and if left as make, a title will be automatically generated. To remove, set to `NULL`.

`sub` String, sets the plot subtitle

`ylab` String, sets the continuous-axis label (=y-axis for box and violin plots, x-axis for ridgeplots). Defaults to "`var`" or "`var` expression" if `var` is a gene.

`y.breaks` Numeric vector, a set of breaks that should be used as major gridlines. c(break1,break2,break3,etc.).

`min, max` Scalars which control the zoom of the plot. These inputs set the minimum / maximum values of the data to show. Default = set based on the limits of the data in var.

`xlab` String which sets the grouping-axis label (=x-axis for box and violin plots, y-axis for ridgeplots). Default is `group.by` so it defaults to the name of the grouping information. Set to `NULL` to remove.

`x.labels` String vector, c("label1","label2","label3",...) which overrides the names of the samples/groups. NOTE: you need to give at least as many labels as there are discrete values in the group.by data.

`x.labels.rotate`
Logical which sets whether the labels should be rotated. Default: `TRUE` for violin and box plots, but `FALSE` for ridgeplots.

`x.reorder` Integer vector. A sequence of numbers, from 1 to the number of groupings, for rearranging the order of x-axis groupings.
Method: Make a first plot without this input. Then, treating the leftmost grouping as index 1, and the rightmost as index n. Values of x.reorder should be these indices, but in the order that you would like them rearranged to be.

`split.nrow, split.ncol`
Integers which set the dimensions of faceting/splitting when a single metadata is given to `split.by`.

`jitter.size` Scalar which sets the size of the jitter shapes.

`jitter.width` Scalar that sets the width/spread of the jitter in the x direction. Ignored in ridgeplots.

`jitter.color` String which sets the color of the jitter shapes

`jitter.shape.legend.size`
Scalar which changes the size of the shape key in the legend. If set to `NA`, `jitter.size` is used.

`jitter.shape.legend.show`
Logical which sets whether the shapes legend will be shown when its shape is determined by `shape.by`.

`boxplot.width` Scalar which sets the width/spread of the boxplot in the x direction

`boxplot.color` String which sets the color of the lines of the boxplot

`boxplot.show.outliers`
Logical, whether outliers should by including in the boxplot. Default is `FALSE` when there is a jitter plotted, `TRUE` if there is no jitter.

`boxplot.fill` Logical, whether the boxplot should be filled in or not. Known bug: when boxplot fill is turned off, outliers do not render.

`vlnplot.lineweight`
Scalar which sets the thickness of the line that outlines the violin plots.

`vlnplot.width` Scalar which sets the width/spread of the jitter in the x direction

`vlnplot.scaling`
String which sets how the widths of the of violin plots are set in relation to eachother. Options are "area", "count", and "width". If the deafult is not right for your data, I recommend trying "width". For a detailed explanation of each, see `geom_violin`.

`ridgeplot.lineweight`
: Scalar which sets the thickness of the ridgeplot outline.

`ridgeplot.scale`
: Scalar which sets the distance/overlap between ridgeplots. A value of 1 means the tallest density curve just touches the baseline of the next higher one. Higher numbers lead to greater overlap. Default = 1.25

`add.line`
: numeric value(s) where one or multiple line should be added

`line.linetype`
: String which sets the type of line for `add.line`. Defaults to "dashed", but any ggplot linetype will work.

`line.color`
: String that sets the color(s) of the `add.line` line(s)

`legend.show`
: Logical. Whether the legend should be displayed. Default = `TRUE`.

`legend.title`
: String or `NULL`, sets the title for the main legend which includes colors and data representations. This input is set to `NULL` by default.

`data.out`
: Logical. When set to `TRUE`, changes the output, from the plot alone, to a list containing the plot (`p`) and data (`data`).

  Note: plotly conversion is turned off in the `data.out = TRUE` setting, but hover.data is still calculated.

`...`
: arguments passed to dittoPlot by dittoRidgePlot, dittoRidgeJitter, and dittoBoxPlot wrappers. Options are all the ones above.

Details

The function creates a dataframe containing the metadata or expression data associated with the given `var` (or if a vector of data is provided, that data). On the discrete axis, data will be grouped by the metadata given to `group.by` and colored by the metadata given to `color.by`. The `assay` and `slot` inputs can be used to change what expression data is used when displaying gene expression. If a set of cells to use is indicated with the `cells.use` input, the data is subset to include only those cells before plotting.

The `plots` argument determines the types of data representation that will be generated, as well as their order from back to front. Options are `"jitter"`, `"boxplot"`, `"vlnplot"`, and `"ridgeplot"`. Inclusion of `"ridgeplot"` overrides boxplot and violin plot and changes the plot to be horizontal.

When `split.by` is provided the name of a metadata containing discrete data, separate plots will be produced representing each of the distinct groupings of the split.by data.

`dittoRidgePlot`, `dittoRidgeJitter`, and `dittoBoxPlot` are included as wrappers of the basic `dittoPlot` function that simply change the default for the `plots` input to be `"ridgeplot"`, `c("ridgeplot","jitter")`, or `c("boxplot","jitter")`, to make such plots even easier to produce.

Value

a ggplot or plotly where continuous data, grouped by sample, age, cluster, etc., shown on either the y-axis by a violin plot, boxplot, and/or jittered points, or on the x-axis by a ridgeplot with or without jittered points.

Alternatively when `data.out=TRUE`, a list containing the plot ("p") and the underlying data as a dataframe ("data").

Alternatively when `do.hover = TRUE`, a plotly converted version of the plot where additional data will be displayed when the cursor is hovered over jitter points.

Functions

- `dittoRidgePlot`: Plots continuous data for cutomizable cells'/samples' groupings horizontally in a density representation
- `dittoRidgeJitter`: dittoRidgePlot, but with jitter overlaid
- `dittoBoxPlot`: Plots continuous data for cutomizable cells'/samples' groupings in boxplot form

Many characteristics of the plot can be adjusted using discrete inputs

- Each data representation has options which are controlled by variables that start with their associated string. For example, all jitter adjustments, like `jitter.size`, start with "`jitter.`".
- Colors can be adjusted with `color.panel`.
- Shapes used in conjunction with `shape.by` can be adjusted with `shape.panel`.
- Titles and axes labels can be adjusted with `main`, `sub`, `xlab`, `ylab`, and `legend.title` arguments.
- The legend can be hidden by setting `legend.show = TRUE`.
- y-axis zoom and tick marks can be adjusted using `min`, `max`, and `y.breaks`.
- x-axis labels and groupings can be changed / reordered using `x.labels` and `x.reorder`, and rotation of these labels can be turned off with `x.labels.rotate = FALSE`.
- Line(s) can be added at single or multiple value(s) by providing these values to `add.line`. Linetype and color are set with `line.linetype`, which is "dashed" by default, and `line.color`, which is "black" by default.
- Single or multiple additional per-cell features can be retrieved and stashed within the underlying data using `extra.vars`. This can be very useful for making manual additional alterations *after* dittoSeq plot generation.

Author(s)

Daniel Bunis

See Also

`multi_dittoPlot` for easy creation of multiple dittoPlots each focusing on a different `var`.

`dittoPlotVarsAcrossGroups` to create dittoPlots that show summarized expression (or values for metadata), accross groups, of multiple `vars` in a single plot.

`dittoRidgePlot`, `dittoRidgeJitter`, and `dittoBoxPlot` for shortcuts to a few 'plots' input shortcuts

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

# Basic dittoplot, with jitter behind a vlnplot (looks better with more cells)
dittoPlot(object = myRNA, var = "gene1", group.by = "timepoint")
```

```
# Color distinctly from the grouping variable using 'color.by'
dittoPlot(object = myRNA, var = "gene1", group.by = "timepoint",
    color.by = "conditions")

# Update the 'plots' input to change / reorder the data representations
dittoPlot(myRNA, "gene1", "timepoint",
    plots = c("vlnplot", "boxplot", "jitter"))

# Modify the look with intuitive inputs
dittoPlot(myRNA, "gene1", "timepoint",
    plots = c("vlnplot", "boxplot", "jitter"),
    boxplot.color = "white",
    main = "CD3E",
    legend.show = FALSE)

# Data can also be split in other ways with 'shape.by' or 'split.by'
dittoPlot(object = myRNA, var = "gene1", group.by = "timepoint",
    plots = c("vlnplot", "boxplot", "jitter"),
    shape.by = "clustering",
    split.by = "SNP") # single split.by element
dittoPlot(object = myRNA, var = "gene1", group.by = "timepoint",
    plots = c("vlnplot", "boxplot", "jitter"),
    split.by = c("groups","SNP")) # row and col split.by elements

# For faceting, instead of using 'split.by', the target data can alternatively
# be given to 'extra.var' to have it added in the underlying dataframe, then
# faceting can be added manually for extra flexibility
dittoPlot(myRNA, "gene1", "clustering",
    plots = c("vlnplot", "boxplot", "jitter"),
    extra.var = "SNP") + facet_wrap("SNP", ncol = 1, strip.position = "left")

# Quickly make a Ridgeplot
dittoRidgePlot(myRNA, "gene1", group.by = "timepoint")

# Quickly make a Boxplot
dittoBoxPlot(myRNA, "gene1", group.by = "timepoint")
```

`dittoPlotVarsAcrossGroups` *Generates a dittoPlot where datapoints are genes/metadata summarizes per groups instead of individual values per cells/samples.*

Description

Generates a dittoPlot where datapoints are genes/metadata summarizes per groups instead of individual values per cells/samples.

Usage

```
dittoPlotVarsAcrossGroups(
  object,
  vars,
```

```
  group.by,
  color.by = group.by,
  summary.fxn = mean,
  cells.use = NULL,
  plots = c("vlnplot", "jitter"),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = "z-score",
  do.hover = FALSE,
  main = NULL,
  sub = NULL,
  ylab = "make",
  y.breaks = NULL,
  min = NULL,
  max = NULL,
  xlab = group.by,
  x.labels = NULL,
  x.labels.rotate = NA,
  x.reorder = NULL,
  color.panel = dittoColors(),
  colors = c(seq_along(color.panel)),
  theme = theme_classic(),
  jitter.size = 1,
  jitter.width = 0.2,
  jitter.color = "black",
  boxplot.width = 0.2,
  boxplot.color = "black",
  boxplot.show.outliers = NA,
  boxplot.fill = TRUE,
  vlnplot.lineweight = 1,
  vlnplot.width = 1,
  vlnplot.scaling = "area",
  ridgeplot.lineweight = 1,
  ridgeplot.scale = 1.25,
  add.line = NULL,
  line.linetype = "dashed",
  line.color = "black",
  legend.show = TRUE,
  legend.title = NULL,
  data.out = FALSE
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object
`vars`	String vector (example: `c("gene1","gene2","gene3")`) which selects which variables, typically genes, to extract from the object, summarize across groups, and add to the plot
`group.by`	String representing the name of a metadata to use for separating the cells/samples into discrete groups.
`color.by`	String representing the name of a metadata to use for setting fills. Great for highlighting subgroups when wanted, but it defaults to `group.by` so this input

can be skipped otherwise. Affects boxplot, vlnplot, and ridgeplot fills.

`summary.fxn` A function which sets how variables' data will be summarized accross the groups. Default is `mean`, which will take the average value, but any function can be used as long as it takes in a numeric vector and returns a single numeric value. Alternative examles: `median`, `max`, `function (x) sum(x!=0)/length(x)`.

`cells.use` String vector of cells'/samples' names which should be included. Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide `USE` in `colnames(object)[USE]` OR `object@cell.names[USE]`.

`plots` String vector which sets the types of plots to include: possibilities = "jitter", "boxplot", "vlnplot", "ridgeplot". Order matters: c("vlnplot", "boxplot", "jitter") will put a violin plot in the back, boxplot in the middle, and then individual dots in the front. See details section for more info.

`assay, slot` single strings or integer that set which data to use when plotting expressin data. See `gene` for more information about how defaults for these are filled in when not provided.

`adjustment` When plotting gene expression (or antibody, or other forms of counts data), should that data be used directly or should it be adjusted to be

- "z-score": DEFAULT, scaled with the scale() function to produce a relative-to-mean z-score representation
- NULL: no adjustment, the normal method for all other ditto expression plotting
- "relative.to.max": divided by the maximum expression value to give percent of max values between [0,1]

`do.hover` Logical. Default = `FALSE`. If set to `TRUE` the object will be converted to a ggplotly object so that data about individual points will be displayed when you hover your cursor over them. The hover data works best for jitter data representations, so it is recommended to have `"jitter"` as the last value of the `plots` input when running using hover.

Note: Currently, incompatible with RidgePlots as plotly does not support the geom.

`main` String which sets the plot title.

`sub` String which sets the plot subtitle.

`ylab` String which sets the y axis label. Default = a combination of then name of the summary function + `adjustment` + "expression". Set to `NULL` to remove.

`y.breaks` Numeric vector, a set of breaks that should be used as major gridlines. c(break1,break2,break3,etc.).

`min, max` Scalars which control the zoom of the plot. These inputs set the minimum / maximum values of the data to show. Default = set based on the limits of the data in var.

`xlab` String which sets the grouping-axis label (=x-axis for box and violin plots, y-axis for ridgeplots). Default is `group.by` so it defaults to the name of the grouping information. Set to `NULL` to remove.

`x.labels` String vector, c("label1","label2","label3",...) which overrides the names of the samples/groups. NOTE: you need to give at least as many labels as there are discrete values in the group.by data.

`x.labels.rotate`

Logical which sets whether the labels should be rotated. Default: `TRUE` for violin and box plots, but `FALSE` for ridgeplots.

`x.reorder` Integer vector. A sequence of numbers, from 1 to the number of groupings, for rearranging the order of x-axis groupings.

Method: Make a first plot without this input. Then, treating the leftmost grouping as index 1, and the rightmost as index n. Values of x.reorder should be these indices, but in the order that you would like them rearranged to be.

`color.panel` String vector which sets the colors to draw from for plot fills. Default = `dittoColors()`.

`colors` Integer vector, the indexes / order, of colors from color.panel to actually use. (Provides an alternative to directly modifying `color.panel`.)

`theme` A ggplot theme which will be applied before dittoSeq adjustments. Default = `theme_classic()`. See https://ggplot2.tidyverse.org/reference/ggtheme.html for other options and ideas.

`jitter.size` Scalar which sets the size of the jitter shapes.

`jitter.width` Scalar that sets the width/spread of the jitter in the x direction. Ignored in ridgeplots.

`jitter.color` String which sets the color of the jitter shapes

`boxplot.width` Scalar which sets the width/spread of the boxplot in the x direction

`boxplot.color` String which sets the color of the lines of the boxplot

`boxplot.show.outliers`
Logical, whether outliers should by including in the boxplot. Default is `FALSE` when there is a jitter plotted, `TRUE` if there is no jitter.

`boxplot.fill` Logical, whether the boxplot should be filled in or not. Known bug: when boxplot fill is turned off, outliers do not render.

`vlnplot.lineweight`
Scalar which sets the thickness of the line that outlines the violin plots.

`vlnplot.width` Scalar which sets the width/spread of the jitter in the x direction

`vlnplot.scaling`
String which sets how the widths of the of violin plots are set in relation to eachother. Options are "area", "count", and "width". If the deafult is not right for your data, I recommend trying "width". For a detailed explanation of each, see `geom_violin`.

`ridgeplot.lineweight`
Scalar which sets the thickness of the ridgeplot outline.

`ridgeplot.scale`
Scalar which sets the distance/overlap between ridgeplots. A value of 1 means the tallest density curve just touches the baseline of the next higher one. Higher numbers lead to greater overlap. Default = 1.25

`add.line` numeric value(s) where one or multiple line should be added

`line.linetype` String which sets the type of line for `add.line`. Defaults to "dashed", but any ggplot linetype will work.

`line.color` String that sets the color(s) of the `add.line` line(s)

`legend.show` Logical. Whether the legend should be displayed. Default = `TRUE`.

`legend.title` String or `NULL`, sets the title for the main legend which includes colors and data representations. This input is set to `NULL` by default.

`data.out` Logical. When set to `TRUE`, changes the output, from the plot alone, to a list containing the plot (`p`) and data (`data`).

Note: plotly conversion is turned off in the `data.out = TRUE` setting, but hover.data is still calculated.

Details

Generally, this function will output a dittoPlot, grouped by sample, age, cluster, etc., where each data point represents the summary (typically `mean`), accross each group, of individual variable's expression, but variables can be genes or metadata.

The data for each element of `vars` is obtained. When elements are genes/features, `assay` and `slot` are utilized to determine which expression data to use, and `adjustment` determines if and how the expression data might be adjusted.

By default, a z-score adjustment is applied to all gene/feature `vars`. Note that this adjustment is applied *before* cells/samples subsetting.

x-axis groupings are then determined using `group.by`, and data for each variable is summarized using the `summary.fxn`.

Finally, data is plotted with the data representation types in `plots`.

Value

a ggplot or plotly where continuous data, grouped by sample, age, cluster, etc., shown on either the y-axis by a violin plot, boxplot, and/or jittered points, or on the x-axis by a ridgeplot with or without jittered points.

Alternatively when `data.out=TRUE`, a list containing the plot ("p") and the underlying data as a dataframe ("data").

Alternatively when `do.hover = TRUE`, a plotly converted version of the plot where additional data will be displayed when the cursor is hovered over jitter points.

Plot Customization

The `plots` argument determines the types of data representation that will be generated, as well as their order from back to front. Options are `"jitter"`, `"boxplot"`, `"vlnplot"`, and `"ridgeplot"`. Each plot type has specific associated options which are controlled by variables that start with their associated string, ex: `jitter.size`.

Inclusion of `"ridgeplot"` overrides boxplot and violin plot and changes the plot to be horizontal.

- Colors can be adjusted with `color.panel`.
- Shapes used in conjunction with `shape.by` can be adjusted with `shape.panel`.
- Titles and axes labels can be adjusted with `main`, `sub`, `xlab`, `ylab`, and `legend.title` arguments.
- The legend can be hidden by setting `legend.show = TRUE`.
- y-axis zoom and tick marks can be adjusted using `min`, `max`, and `y.breaks`.
- x-axis labels and groupings can be changed / reordered using `x.labels` and `x.reorder`, and rotation of these labels can be turned off with `x.labels.rotate = FALSE`.
- Line(s) can be added at single or multiple value(s) by providing these values to `add.line`. Linetype and color are set with `line.linetype`, which is "dashed" by default, and `line.color`, which is "black" by default.

Author(s)

Daniel Bunis

See Also

dittoPlot and multi_dittoPlot for plotting of single or mutliple expression and metadata vars, each as separate plots, on a per cell/sample basis.

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

##########
### Generate some random data
##########
# Zero-inflated Expression
nsamples <- 60
exp <- rpois(1000*nsamples, 20)
exp[sample(c(TRUE,TRUE,FALSE),1000*nsamples, TRUE)] <- 0
exp <- matrix(exp, ncol=nsamples)
colnames(exp) <- paste0("sample", seq_len(ncol(exp)))
rownames(exp) <- paste0("gene", seq_len(nrow(exp)))
logexp <- log2(exp + 1)

# Metadata
conds <- factor(rep(c("condition1", "condition2"), each=nsamples/2))
timept <- rep(c("d0", "d3", "d6", "d9"), each = 15)
genome <- rep(c(rep(TRUE,7),rep(FALSE,8)), 4)
grps <- sample(c("A","B","C","D"), nsamples, TRUE)

# We can add these directly during import, or after.
myscRNA <- importDittoBulk(x = list(counts = exp, logcounts = logexp),
    metadata = data.frame(conditions = conds, timepoint = timept,
        SNP = genome, groups = grps))

# Pick a set of genes
genes <- getGenes(myscRNA)[1:30]

dittoPlotVarsAcrossGroups(
    myscRNA, genes, group.by = "timepoint")

# Color can be controlled separately from grouping with 'color.by'
#   Just note: all groupings must map to a single color.
dittoPlotVarsAcrossGroups(myscRNA, genes, "timepoint",
    color.by = "conditions")

# To change it to have the violin plot in the back, a jitter on
#  top of that, and a white boxplot with no fill in front:
dittoPlotVarsAcrossGroups(myscRNA, genes, "timepoint", "conditions",
    plots = c("vlnplot","jitter","boxplot"),
    boxplot.color = "white", boxplot.fill = FALSE)

## Data can be summaryized in other ways by changing the summary.fxn input.
#  Often, it makes sense to turn off the z-score adjustment in such cases.
#  median
dittoPlotVarsAcrossGroups(myscRNA, genes, "timepoint", "conditions",
    summary.fxn = median,
```

```
    adjustment = NULL)
#  Percent non-zero expression
percent <- function(x) {sum(x!=0)/length(x)}
dittoPlotVarsAcrossGroups(myscRNA, genes, "timepoint", "conditions",
    summary.fxn = percent,
    adjustment = NULL)

# To investigate the identities of outlier genes, we can turn on hovering
# (if the plotly package is available)
if (requireNamespace("plotly", quietly = TRUE)) {
    dittoPlotVarsAcrossGroups(
        myscRNA, genes, "timepoint", "conditions",
        do.hover = TRUE)
}
```

`dittoScatterPlot` *Show RNAseq data overlayed on a scatter plot*

Description

Show RNAseq data overlayed on a scatter plot

Usage

```
dittoScatterPlot(
  object,
  x.var,
  y.var,
  color.var = NULL,
  shape.by = NULL,
  split.by = NULL,
  extra.vars = NULL,
  cells.use = NULL,
  show.others = FALSE,
  size = 1,
  opacity = 1,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  split.nrow = NULL,
  split.ncol = NULL,
  assay.x = .default_assay(object),
  slot.x = .default_slot(object),
  adjustment.x = NULL,
  assay.y = .default_assay(object),
  slot.y = .default_slot(object),
  adjustment.y = NULL,
  assay.color = .default_assay(object),
  slot.color = .default_slot(object),
  adjustment.color = NULL,
  assay.extra = .default_assay(object),
  slot.extra = .default_slot(object),
```

```
  adjustment.extra = NULL,
  do.hover = FALSE,
  hover.data = NULL,
  hover.assay = .default_assay(object),
  hover.slot = .default_slot(object),
  hover.adjustment = NULL,
  shape.panel = c(16, 15, 17, 23, 25, 8),
  rename.color.groups = NULL,
  rename.shape.groups = NULL,
  min.color = "#F0E442",
  max.color = "#0072B2",
  min = NULL,
  max = NULL,
  xlab = x.var,
  ylab = y.var,
  main = "make",
  sub = NULL,
  theme = theme_bw(),
  legend.show = TRUE,
  legend.color.title = color.var,
  legend.color.size = 5,
  legend.color.breaks = waiver(),
  legend.color.breaks.labels = waiver(),
  legend.shape.title = shape.by,
  legend.shape.size = 5,
  data.out = FALSE
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object
`x.var, y.var`	Single string giving a gene or metadata that will be used for the x- and y-axis of the scatterplot. Note: must be continuous. Alternatively, can be a directly supplied numeric vector of length equal to the total number of cells/samples in `object`.
`color.var`	Single string giving a gene or metadata that will set the color of cells/samples in the plot. Alternatively, can be a directly supplied numeric or string, vector or a factor of length equal to the total number of cells/samples in `object`.
`shape.by`	Single string giving a metadata (Note: must be discrete.) that will set the shape of cells/samples in the plot. Alternatively, can be a directly supplied string vector or a factor of length equal to the total number of cells/samples in `object`.
`split.by`	1 or 2 strings naming discrete metadata to use for splitting the cells/samples into multiple plots with ggplot faceting. When 2 metadatas are named, c(row,col), the first is used as rows and the second is used for columns of the resulting grid. When 1 metadata is named, shape control can be achieved with `split.nrow` and `split.ncol`
`extra.vars`	String vector providing names of any extra metadata to be stashed in the dataframe supplied to `ggplot(data)`.

Useful for making custom alterations *after* dittoSeq plot generation.

`cells.use` String vector of cells'/samples' names which should be included.

Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide `USE` in `object@cell.names[USE]` OR `colnames(object)[USE]`).

`show.others` Logical. TRUE by default, whether other cells should be shown in the background in light gray.

`size` Number which sets the size of data points. Default = 1.

`opacity` Number between 0 and 1. Great for when you have MANY overlapping points, this sets how solid the points should be: 1 = not see-through at all. 0 = invisible. Default = 1. (In terms of typical ggplot variables, = alpha)

`color.panel` String vector which sets the colors to draw from. `dittoColors()` by default, see `dittoColors` for contents.

`colors` Integer vector, the indexes / order, of colors from color.panel to actually use

`split.nrow, split.ncol`

Integers which set the dimensions of faceting/splitting when a single metadata is given to `split.by`.

`assay.x, assay.y, assay.color, assay.extra, slot.x, slot.y, slot.color, slot.extra, adjustment.x, ad`

assay, slot, and adjustment set which data to use when the axes, coloring, or `extra.vars` are based on expression/counts data. See `gene` for additional information.

`do.hover` Logical which controls whether the object will be converted to a plotly object so that data about individual points will be displayed when you hover your cursor over them. `hover.data` argument is used to determine what data to use.

`hover.data` String vector of gene and metadata names, example: `c("meta1","gene1","meta2","gene2")` which determines what data to show on hover when `do.hover` is set to `TRUE`.

`hover.assay, hover.slot, hover.adjustment`

Similar to the x, y, color, and extra versions, when showing expression data upon hover, these set what data will be shown.

`shape.panel` Vector of integers corresponding to ggplot shapes which sets what shapes to use. When discrete groupings are supplied by `shape.by`, this sets the panel of shapes. When nothing is supplied to `shape.by`, only the first value is used. Default is a set of 6, `c(16,15,17,23,25,8)`, the first being a simple, solid, circle.

Note: Unfortunately, shapes can be hard to see when points are on top of each other & they are more slowly processed by the brain. For these reasons, even as a color blind person myself writing this code, I recommend use of colors for variables with many discrete values.

`rename.color.groups, rename.shape.groups`

String vector containing new names for the identities of the color or shape overlay groups.

`min.color` color for lowest values of var/min. Default = yellow

`max.color` color for highest values of var/max. Default = blue

`min, max` Numbers which set the values associated with the minimum and maximum colors.

`xlab, ylab` Strings which set the labels for the axes. To remove, set to `NULL`.

`main` String, sets the plot title. A default title is automatically generated if based on `color.var` and `shape.by` when either are provided. To remove, set to `NULL`.

sub — String, sets the plot subtitle.

theme — A ggplot theme which will be applied before dittoSeq adjustments. Default = theme_bw(). See https://ggplot2.tidyverse.org/reference/ggtheme.html for other options and ideas.

legend.show — Logical. Whether any legend should be displayed. Default = TRUE.

legend.color.title, legend.shape.title — Strings which set the title for the color or shape legends.

legend.color.size, legend.shape.size — Numbers representing the size at which shapes should be plotted in the color and shape legends (for discrete variable plotting). Default = 5. *Enlarging the icons in the colors legend is incredibly helpful for making colors more distinguishable by color blind individuals.

legend.color.breaks — Numeric vector which sets the discrete values to show in the color-scale legend for continuous data.

legend.color.breaks.labels — String vector, with same length as legend.breaks, which renames what's displayed next to the tick marks of the color-scale.

data.out — Logical. When set to TRUE, changes the output, from the plot alone, to a list containing the plot ("p"), a data.frame containing the underlying data for target cells ("Target_data"), and a data.frame containing the underlying data for non-target cells ("Others_data").

Note: do.hover plotly conversion is turned off in this setting, but hover.data is still calculated.

Details

This function creates a dataframe with X, Y, color, shape, and faceting data determined by x.var, y.var, color.var, shape.var, and split.by. Any extra gene or metadata requested with extra.var is added as well. For expression/counts data, assay, slot, and adjustment inputs (.x, .y, and .color) can be used to change which data is used, and if it should be adjusted in some way.

Next, if a set of cells or samples to use is indicated with the cells.use input, then the dataframe is split into Target_data and Others_data based on subsetting by the target cells/samples.

Finally, a scatter plot is created using these dataframes. Non-target cells are colored in gray if show.others=TRUE, and target cell data is displayed on top, colored and shaped based on the color.var- and shape.by-associated data. If split.by was used, the plot will be split into a matrix of panels based on the associated groupings.

Value

a ggplot scatterplot where colored dots and/or shapes represent individual cells/samples. X and Y axes can be gene expression, numeric metadata, or manually supplied values.

Alternatively, if data.out=TRUE, a list containing three slots is output: the plot (named 'p'), a data.table containing the underlying data for target cells (named 'Target_data'), and a data.table containing the underlying data for non-target cells (named 'Others_data').

Alternatively, if do.hover is set to TRUE, the plot is coverted from ggplot to plotly & cell/sample information, determined by the hover.data input, is retrieved, added to the dataframe, and displayed upon hovering the cursor over the plot.

Many characteristics of the plot can be adjusted using discrete inputs

- `size` and `opacity` can be used to adjust the size and transparency of the data points.
- Colors used can be adjusted with `color.panel` and/or `colors` for discrete data, or `min`, `max`, `min.color`, and `max.color` for continuous data.
- Shapes used can be adjusted with `shape.panel`.
- Color and shape labels can be changed using `rename.color.groups` and `rename.shape.groups`.
- Titles and axes labels can be adjusted with `main`, `sub`, `xlab`, `ylab`, and `legend.title` arguments.
- Legends can also be adjusted in other ways, using variables that all start with "`legend.`" for easy tab completion lookup.

Author(s)

Daniel Bunis

See Also

`getGenes` and `getMetas` to see what the `x.var`, `y.var`, `color.var`, `shape.by`, and `hover.data` options are.

`dittoDimPlot` for making very similar data representations, but where dimensionality reduction (PCA, t-SNE, UMAP, etc.) dimensions are the scatterplot axes.

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

# Mock up some nCount_RNA and nFeature_RNA metadata
#  == the default way to extract
myRNA$nCount_RNA <- runif(60,200,1000)
myRNA$nFeature_RNA <- myRNA$nCount_RNA*runif(60,0.95,1.05)
# and also percent.mito metadata
myRNA$percent.mito <- sample(c(runif(50,0,0.05),runif(10,0.05,0.2)))

dittoScatterPlot(
    myRNA, x.var = "nCount_RNA", y.var = "nFeature_RNA")

# Shapes or colors can be overlaid representing discrete metadata
#   or (only colors) continuous metadata / expression data by providing
#   metadata or gene names to 'color.var' and 'shape.by'
dittoScatterPlot(
    myRNA, x.var = "nCount_RNA", y.var = "nFeature_RNA",
    color.var = "percent.mito")
dittoScatterPlot(
    myRNA, x.var = "gene1", y.var = "gene2",
    color.var = "groups",
    shape.by = "SNP",
    size = 3)
```

```
dittoScatterPlot(
    myRNA, x.var = "gene1", y.var = "gene2",
    color.var = "gene3")

# Data can be "split" or faceted by a discrete variable as well.
dittoScatterPlot(
    myRNA, x.var = "gene1",
    y.var = "gene2",
    split.by = "timepoint") # single split.by element
dittoScatterPlot(
    myRNA, x.var = "gene1",
    y.var = "gene2",
    split.by = c("groups","SNP")) # row and col split.by elements
# OR with 'extra.vars' plus manually faceting for added control
dittoDimPlot(myRNA, "gene1",
    extra.vars = c("SNP")) +
    facet_wrap("SNP", ncol = 1, strip.position = "left")

# Note: scatterplots like this can be very useful for dataset QC, especially
#   with percentage of reads coming from genes as the color overlay.
```

---

dittoSeq *dittoSeq*

Description

This package was built to make the analysis and visualization of single-cell and bulk RNA-sequencing data accessible for both experience and novice coders, and for colorblind individuals.

Details

Includes many plotting functions (dittoPlot, dittoDimPlot, dittoBarPlot, dittoHeatmap, ...), color adjustment functions (Simulate, Darken, Lighten), and helper funtions (meta, gene, isMeta, getMetas, ...) to aid in making sense of single cell or bulk RNA sequencing data. All included plotting functions produce a ggplot (or plotly, or pheatmap for dittoHeatmap) and can spit out full plot with just a few arguments. Many additional arguments are available for customization to generate complex publication-ready figures.

Default color panel is colorblind friendly [Wong B, "Points of view: Color blindness." Nature Methods, 2011.](https://www.nature.com/articles/nmeth.1618).

For more information, to give feedback, or to suggest new features, see the github, [here](https://github.com/dtm2451/Dit

Author(s)

Daniel Bunis

gene *Returns the expression values of a gene for all cells/samples*

Description

Returns the expression values of a gene for all cells/samples

Usage

```
gene(
  gene,
  object,
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL
)
```

Arguments

gene	quoted "gene" name = REQUIRED. the gene whose expression data should be retrieved.
object	A target Seurat or SingleCellExperiment object
assay, slot	single strings or integer that set which data to use. Seurat and SingleCellExperiments deal with these differently, so be sure to check the documentation for whichever object you are using. When not provided, these typical defaults for the provided `object` class are used: • SingleCellExperiment (single-cell or bulk data): `assay` = "logcounts", "normcounts", "counts", or the first element of assays(object), `slot` not used • Seurat-v3: `assay` = DefaultAssay(object), `slot` = "data" • Seurat-v2: `assay` not used, `slot` = "data"
adjustment	Should expression data be used directly (default) or should it be adjusted to be • "z-score": scaled with the scale() function to produce a relative-to-mean z-score representation • "relative.to.max": divided by the maximum expression value to give percent of max values between [0,1]

Value

Returns the expression values of a gene for all cells/samples.

Author(s)

Daniel Bunis

Examples

```
example(importDittoBulk, echo = FALSE)
gene("gene1", object = myRNA, assay = "counts")

# z-scored
gene("gene1", object = myRNA, assay = "counts", adjustment = "z-score")

# To see expression of the gene for the default assay that dittoSeq would use
# leave out the assay input
# (For this object, the default assay is the logcounts assay)
gene("gene1", myRNA)

# Seurat (raw counts)
if (!requireNamespace("Seurat")) {
    gene("CD14", object = Seurat::pbmc, assay = "RNA", slot = "counts")
}
```

---

getGenes *Returns the names of all genes of a target object.*

Description

Returns the names of all genes of a target object.

Usage

```
getGenes(object, assay = .default_assay(object))
```

Arguments

object	A target Seurat or SingleCellExperiment object
assay	single string or integer that sets which set of seq data inside the object to check.

Value

A string vector, returns the names of all genes of the object for the requested assay.

Author(s)

Daniel Bunis

See Also

isGene for returning all genes in an object

gene for obtaining the expression data of genes

Examples

```
example(importDittoBulk, echo = FALSE)
getGenes(object = myRNA, assay = "counts")

# To see all genes of an object for the default assay that dittoSeq would use
# leave out the assay input
getGenes(myRNA)

# Seurat
# pbmc <- Seurat::pbmc_small
# # To see all genes of an object of a particular assay
# getGenes(pbmc, assay = "RNA")
```

`getMetas` *Returns the names of all meta.data slots of a target object.*

Description

Returns the names of all meta.data slots of a target object.

Usage

```
getMetas(object, names.only = TRUE)
```

Arguments

`object`	A target Seurat or SingleCellExperiment object
`names.only`	Logical, `TRUE` by default, which sets whether just the names should be output versus the entire metadata dataframe.

Value

A string vector of the names of all metadata slots of the `object`, or alternatively the entire dataframe of metadatas if `names.only` is set to `FALSE`

Author(s)

Daniel Bunis

See Also

`isMeta` for checking if certain metadata slots exist in an `object`

`meta` for obtaining the contants of metadata slots

Examples

```
example(importDittoBulk, echo = FALSE)

# To see all metadata slots of an object
getMetas(myRNA)

# To retrieve the entire metadata matrix
getMetas(myRNA, names.only = FALSE)
```

`getReductions` *Returns the names of all dimensionality reduction slots of a target object.*

Description

Returns the names of all dimensionality reduction slots of a target object.

Usage

```
getReductions(object)
```

Arguments

`object`	A target Seurat or SingleCellExperiment object

Value

A string vector of the names of all dimensionality reduction slots of the `object`. These represent the options for the `reduction.use` input of `dittoDimPlot`.

Author(s)

Daniel Bunis

Examples

```
example("addDimReduction", echo = FALSE)

# To see all metadata slots of an object
getReductions(myRNA)
```

`importDemux` *Extracts Demuxlet information into a pre-made SingleCellExperiment or Seurat object*

Description

Extracts Demuxlet information into a pre-made SingleCellExperiment or Seurat object

Usage

```
importDemux(
  object,
  raw.cell.names = NULL,
  lane.meta = NULL,
  lane.names = NA,
  demuxlet.best,
  trim.before_ = TRUE,
  bypass.check = FALSE,
  verbose = TRUE
)
```

Arguments

`object`	A pre-made Seurat(v3+) or SingleCellExperiment object to add demuxlet information to.
`raw.cell.names`	A string vector consisting of the raw cell barcodes of the object as they would have been output by cellranger aggr. Format per cell.name = NNN...NNN-# where NNN...NNN are the cell barcode nucleotides, and # is the lane number. This input should be used when additional information has been added directly into the cell names outside of Seurat's standard merge prefix: "user-text_".
`lane.meta`	A string which names a metadata slot that contains which cells came from which droplet-generation wells.
`lane.names`	String vector which sets how the lanes should be named (if you want to give them something different from the default = Lane1, Lane2, Lane3...)
`demuxlet.best`	String or String vector pointing to the location(s) of the .best output file from running of demuxlet. Alternatively, a data.frame representing an already imported .best matrix.
`trim.before_`	Logical which sets whether any characters in front of an "_" should be deleted from the `raw.cell.names` before matching with demuxlet barcodes.
`bypass.check`	Logical which sets whether the function should run even when meta.data slots would be over-written.
`verbose`	whether to print messages about the stage of this process that is currently being run & also the summary at the end.

Details

The function takes in a previously generated Seurat or SingleCellExperiment object. It also takes in demuxlet information either in the form of

1: the location of a single demuxlet.best out file,

2: the locations of multiple demuxlet.best output files,

or 3: a user-constructed data.frame created by reading in a demuxlet.best file.

If a metadata slot name is provided to `lane.meta`, information in that metadata slot is copied into a metadata slot called "Lane". Alternatively, if `lane.meta` is left as `NULL`, separate lanes are assumed to be marked by distinct values of "-#", as is the typical output of the 10X cellranger count & aggr pipeline. In these situations, the `lane.names` input can be used to set specific names for each lane. "Lane1", "Lane2", "Lane3", etc, are used b y default.

The `colnames(object)` are used by default, but if these have been modified from what would have been given to demuxlet, outside of "-#" at the end or "***_" as can be added in common merge functions, you can alternatively provide `raw.cell.names`.

Barcodes in the demuxlet data are matched to barcodes in the `object` and then singlet/doublet/ambiguous calls and identities are parsed and carried into metadata. (When demuxlet information is provided as a set of separate files (recommended for use with cellranger aggr), the "-#" at the ends of barcodes in these files are incremented on read-in so that they can match the incrementation applied by cellranger aggr. See note on multi-well 10X data below for more.)

Finally, a summary of the results including mean number of SNPs and percentages of singlets and doublets is output unless `verbose` is set to `FALSE`.

Lane information and demuxlet calls and statistics are imported into the `object` as these metadata:

- Lane = guided by `lane.meta` import input or "-#"s in barcodes, represents the separate droplet-generation lanes.
- Sample = The sample call, parsed from the BEST column
- demux.doublet.call = whether the sample was a singlet (SNG), doublet (DBL), or ambiguious (AMB), parsed from the BEST column
- demux.RD.TOTL = RD.TOTL column
- demux.RD.PASS = RD.PASS column
- demux.RD.UNIQ = RD.UNIQ column
- demux.N.SNP = N.SNP column
- demux.PRB.DBL = PRB.DBL column
- demux.barcode.dup = (Only generated when TRUEs will exist) whether a cell's barcode in the demuxlet.best refered to only 1 cell in the `object`. (When TRUE, indicates that cells from distinct lanes were interpretted together by demuxlet. These will often be mistakenly called as doublets.)

Note: "-#" information added by cellranger functions is not removed. Doing so would cause cells, from separate 10X wells, which ended up with similar barcodes to become indistinguishable. In demuxlet itself, ignorance of lane information leads to artificial doublet calls. In `importDemux`, ignorance of lane information can lead to import of improper demuxlet annotations. For this reason, `importDemux` checks for whether such artificial duplicates likely happened. See the recommended cellranger/demuxlet pipeline below for specific suggestions for how to use this function with multi-well 10X data.

Value

The Seurat or SingleCellExperiment object with metadata added for "Sample" calls and other relevant statistics.

For multi-well 10X data

10X recommends running cellranger counts individually for each well/lane. This leads to creation of separate genes x cells counts matrices for each lane. *Demuxlet should also be run separately for each lane in order to minimize the informatic generation of artificial doublets. Afterwards, there are many common methods of importing/merging such multi-well 10X data into a single object in R. Technical differences: All options will alter the cell barcode names in a way that makes them unique across lanes, but how they do can be different. Technical issue: Neither method adjusts the bacode names that are embedded within the BAM files which a user must supply to Demuxlet, so that data needs to be modified in a proper way in order to make the `object` cellnames and demuxlet BARCODEs match.

`importDemux` is built for working with the cellranger aggr barcodes output, but can be used for demuxlet datasets processed differently as well.

- Option 1: merging matrices of all lanes with cellranger aggr before R import. Barcode uniquification method: A "-1", "-2", "-3", ... "-#" is appended to the end of all barcode names. The number is incremented for each succesive lane. (Note: lane-numbers depend on the order in which they were supplied to cellranger aggr.)
- Option 2: Importing into Seurat or SingleCellExperiment, then merging these objects. Barcode uniquifiction method: user-defined strings are appended to the start of the barcodes, followed by an "_", for Seurat merge, and importDemux will ignore these. Alternatively, consistent barcodes can be supplied separately to the `raw.cell.names` input.

The fix: `importDemux` ignores all information before a "_" in cellnames when `trim.before_` is left as TRUE, but utilizes the "-#" information at the ends of Seurat cellnames.

- Option 1: `importDemux` can adjust the "-#" in the Demuxlet BARCODEs automatically for users before performing the matching step. In order to take advantage of the automatic barcodes adjustment, just supply a vector containing the locations of the sepearate .best outputs for each lane, in the same order that lanes were combined in cellranger aggr.
- Option 2: To use with this method, it's easiest to run `importDemux` on each lane's Seurat or SingleCellExperiment object separately & provide a unique name for each lane to the `lane.names` input, BEFORE merging into a single Seurat object.

Run in these ways, demuxlet information can be matched to proper cells, and lane assignments can be properly reported in the "Lane" metadata slot.

Author(s)

Daniel Bunis

See Also

Included QC visualizations:

`demux.calls.summary` for plotting the number of sample annotations assigned within each lane.

`demux.SNP.summary` for plotting the number of SNPs measured per cell.

Or, see Kang et al. Nature Biotechnology, 2018 https://www.nature.com/articles/nbt.4042 for more information about the demuxlet cell-sample deconvolution method.

Examples

```
#Prep: loading in an example dataset and sample demuxlet data
example("importDittoBulk", echo = FALSE)
demux <- demuxlet.example
colnames(myRNA) <- demux$BARCODE[seq_len(ncol(myRNA))]

###
### Method 1: Lanes info stored in a metadata
###

# Notice there is a groups metadata in this Seurat object.
getMetas(myRNA)
# We will treat these as if that holds Lane information

# Now, running importDemux:
myRNA <- importDemux(
    myRNA,
    lane.meta = "groups",
    demuxlet.best = demux)

# Note, importDemux can also take in the location of the .best file.
#   myRNA <- importDemux(
#       object = myRNA,
#       lane.meta = "groups",
#       demuxlet.best = "Location/filename.best")

# demux.SNP.summary() and demux.calls.summary() can now be used.
demux.SNP.summary(myRNA)
demux.calls.summary(myRNA)

###
### Method 2: cellranger aggr combined data (denoted with "-#" in barcodes)
###

# If cellranger aggr was used, lanes will be denoted by "-1", "-2", ... "-#"
#   at the ends of Seurat cellnames.
# Demuxlet should be run on each lane individually.
# Provided locations of each demuxlet.best output file, *in the same order
#   that lanes were provided to cellranger aggr* this function will then
#   adjust the "-#" within the .best BARCODEs automatically before matching
#
# myRNA <- importDemux(
#     object = myRNA,
#     demuxlet.best = c(
#         "Location/filename1.best",
#         "Location/filename2.best"),
#     lane.names = c("g1","g2"))
```

`importDittoBulk` *import bulk sequencing data into a format that dittoSeq functions expect.*

Description

import bulk sequencing data into a format that dittoSeq functions expect.

Usage

```
importDittoBulk(x, ...)

## S4 method for signature 'SummarizedExperiment'
importDittoBulk(x, reductions = NULL, metadata = NULL, combine_metadata = TRUE)

## S4 method for signature 'DGEList'
importDittoBulk(x, reductions = NULL, metadata = NULL, combine_metadata = TRUE)

## S4 method for signature 'list'
importDittoBulk(x, reductions = NULL, metadata = NULL)
```

Arguments

x	a DGEList, or SummarizedExperiment (includes DESeqDataSet) class object containing the sequencing data to be imported
...	For the generic, additional arguments passed to specific methods.
reductions	a named list of dimensionality reduction embeddings matrices. names will become the names of the dimensionality reductions and how each will be used with the reduction.use input of dittoDimPlot rows of the matrices should represent the different cells/samples of the dataset, and columns the different dimensions
metadata	a data.frame like object containing columns of extra information about the cells/samples (rows). The names of these columns can then be used to tretrieve and plot such data in any dittoSeq visualizations.
combine_metadata	Logical which sets whether original colData (DESeqDataSet/SummarizedExperiment) or $samples (DGEList) from x should be retained.

Value

A SingleCellExperiment object containing all assays (DESeqDataSet or SummarizeedExperiment) or all common slots (DGEList) of the input x, as well as any dimensionality reductions provided to reductions, and any provided metadata stored in colData.

When combine_metadata is set to FALSE, metadata inside x (colData or $samples) is ignored entirely. When combine_metadata is TRUE (the default), metadata inside x is combined with what is provided to the metadata input; but names must be unique, so when there are similarly named slots, the values provided to the metadata input are used.

Note

One recommended assay to create if it is not already present in your dataset, is a log-normalized version of the counts data. The logNormCounts function of the scater package is an easy way to make such a slot. dittoSeq defaults to grabbing expression data from an assay named logcounts > normcounts > counts

See Also

SingleCellExperiment for more information about this storage system.

Examples

```
## Bulk data is stored as a SingleCellExperiment
library(SingleCellExperiment)

# Generate some random data
nsamples <- 60
exp <- matrix(rpois(1000*nsamples, 20), ncol=nsamples)
colnames(exp) <- paste0("sample", seq_len(ncol(exp)))
rownames(exp) <- paste0("gene", seq_len(nrow(exp)))
logexp <- log2(exp + 1)

# Dimensionality Reductions
pca <- matrix(runif(nsamples*5,-2,2), nsamples)
tsne <- matrix(rnorm(nsamples*2), nsamples)

# Some Metadata
conds <- factor(rep(c("condition1", "condition2"), each=nsamples/2))
timept <- rep(c("d0", "d3", "d6", "d9"), each = 15)
genome <- rep(c(rep(TRUE,7),rep(FALSE,8)), 4)
grps <- sample(c("A","B","C","D"), nsamples, TRUE)
clusts <- as.character(1*(tsne[,1]>0&tsne[,2]>0) +
                       2*(tsne[,1]<0&tsne[,2]>0) +
                       3*(tsne[,1]>0&tsne[,2]<0) +
                       4*(tsne[,1]<0&tsne[,2]<0))

### We can import the counts directly, or as a SummarizedExperiment
myRNA <- importDittoBulk(
    x = list(counts = exp,
         logcounts = logexp))

### Adding metadata & PCA or other dimensionality reductions
# We can add these directly during import, or after.
myRNA <- importDittoBulk(
    x = list(counts = exp,
        logcounts = logexp),
    metadata = data.frame(
        conditions = conds,
        timepoint = timept,
        SNP = genome,
        groups = grps),
    reductions = list(
        pca = pca))

myRNA$clustering <- clusts

myRNA <- addDimReduction(
    myRNA,
    embeddings = tsne,
    name = "tsne")

# (other packages SCE manipulations can also be used)
```

```
### When we import from SummarizedExperiment, all metadata is retained.
# The object is just 'upgraded' to hold extra slots.
# The input is the same, aside from a message when metadata are replaced.
se <- SummarizedExperiment(
    list(counts = exp, logcounts = logexp))
myRNA <- importDittoBulk(
    x = se,
    metadata = data.frame(
        conditions = conds,
        timepoint = timept,
        SNP = genome,
        groups = grps,
        clustering = clusts),
    reductions = list(
        pca = pca,
        tsne = tsne))
myRNA

### For DESeq2, how we might have made this:
# DESeqDataSets are SummarizedExperiments, and behave similarly
# library(DESeq2)
# dds <- DESeqDataSetFromMatrix(
#     exp, data.frame(conditions), ~ conditions)
# dds <- DESeq(dds)
# dds_ditto <- importDittoBulk(dds)

### For edgeR, DGELists are a separate beast.
# dittoSeq imports what I know to commonly be inside them, but please submit
# an issue on the github (dtm2451/dittoSeq) if more should be retained.
# library(edgeR)
# dgelist <- DGEList(counts=exp, group=conditions)
# dge_ditto <- importDittoBulk(dgelist)
```

`isBulk` *Retrieve whether a SingleCellObject should be treated as single-cell versus bulk*

Description

Retrieve whether a SingleCellObject should be treated as single-cell versus bulk

Usage

```
isBulk(object)
```

Arguments

`object`	A target SingleCellExperiment object Alternatively, anything else, but then the result will always be `FALSE`

Value

Logical: whether the provided object would be treated as bulk data by dittoSeq

Examples

```
example(importDittoBulk, echo = FALSE)
myRNA

isBulk(myRNA)
```

`isGene` *Tests if input is the name of a gene in a target object.*

Description

Tests if input is the name of a gene in a target object.

Usage

```
isGene(test, object, assay = .default_assay(object), return.values = FALSE)
```

Arguments

`test`	String or vector of strings, the "potential.gene.name"(s) to check for.
`object`	A target Seurat or SingleCellExperiment object
`assay`	single string or integer that sets which set of seq data inside the object to check.
`return.values`	Logical which sets whether the function returns a logical `TRUE`/`FALSE` versus the `TRUE` `test` values . Default = `FALSE` REQUIRED, unless 'DEFAULT <-"object"' has been run.

Value

Returns a logical vector indicating whether each instance in `test` is a rowname within the requested `assay` of the `object`. Alternatively, returns the values of `test` that were indeed rownames if `return.values` = `TRUE`.

Author(s)

Daniel Bunis

See Also

`getGenes` for returning all genes in an `object`

`gene` for obtaining the expression data of genes

Examples

```
example(importDittoBulk, echo = FALSE)

# To see the first 10 genes of an object of a particular assay
getGenes(myRNA, assay = "counts")[1:10]

# To see all genes of an object for the default assay that dittoSeq would use
# leave out the assay input (again, remove `head()`)
head(getGenes(myRNA))
```

```
# To test if something is a gene in an object:
isGene("gene1", object = myRNA) # TRUE
isGene("CD12345", myRNA) # FALSE

# To test if many things are genes of an object
isGene(c("gene1", "gene2", "not-a-gene", "CD12345"), myRNA)

# 'return.values' input is especially useful in these cases.
isGene(c("gene1", "gene2", "not-a-gene", "CD12345"), myRNA,
    return.values = TRUE)
```

`isMeta` *Tests if an input is the name of a meta.data slot in a target object.*

Description

Tests if an input is the name of a meta.data slot in a target object.

Usage

```
isMeta(test, object, return.values = FALSE)
```

Arguments

`test`	String or vector of strings, the "potential.metadata.name"(s) to check for.
`object`	A target Seurat or SingleCellExperiment object
`return.values`	Logical which sets whether the function returns a logical `TRUE`/`FALSE` versus the `TRUE` `test` values . Default = `FALSE`

Details

For Seurat objects, also returns TRUE for the input `"ident"` because, for all dittoSeq visualiztions, `"ident"` will retrieve a Seurat objects' clustering slot.

Value

Returns a logical or logical vector indicating whether each instance in `test` is a meta.data slot within the `object`. Alternatively, returns the values of `test` that were indeed metadata slots if `return.values` = `TRUE`.

Author(s)

Daniel Bunis

See Also

`getMetas` for returning all metadata slots of an `object`

`meta` for obtaining the contants of metadata slots

Examples

```
example(importDittoBulk, echo = FALSE)

# To check if something is a metadata slot
isMeta("timepoint", object = myRNA) # FTRUE
isMeta("nCount_RNA", object = myRNA) # FALSE

# To test if many things are metadata of an object
isMeta(c("age","groups"), myRNA) # FALSE, TRUE

# 'return.values' input is especially useful in these cases.
isMeta(c("age","groups"), myRNA,
    return.values = TRUE)

# Alternatively, to see all metadata slots of an object, use getMetas
getMetas(myRNA)
```

Lighten *Lightens input colors by a set amount*

Description

A wrapper for the lighten function of the colorspace package.

Usage

```
Lighten(colors, percent.change = 0.25, relative = TRUE)
```

Arguments

colors	the color(s) input. Can be a list of colors, for example, /codedittoColors().
percent.change	# between 0 and 1. the percentage to darken by. Defaults to 0.25 if not given.
relative	TRUE/FALSE. Whether the percentage should be a relative change versus an absolute one. Default = TRUE.

Value

Return a lighter version of the color in hexadecimal color form (="#RRGGBB" in base 16)

Author(s)

Daniel Bunis

Examples

```
Lighten("blue") #"blue" = "#0000FF"
#Output: "#4040FF"
Lighten(dittoColors()[1:8]) #Works for multiple color inputs as well.
```

`meta` *Returns the values of a meta.data for all cells/samples*

Description

Returns the values of a meta.data for all cells/samples

Usage

```
meta(meta, object)
```

Arguments

`meta`	String, the name of the "metadata" slot to grab. OR "ident" to retireve the clustering of a Seurat `object`.
`object`	A target Seurat or SingleCellExperiment object

Value

A named vector. Returns the values of a metadata slot, or the clustering slot if `meta = "ident"` and the `object` is a Seurat. Names of values will be the cell/sample names.

Author(s)

Daniel Bunis

See Also

`metaLevels` for returning just the unique discrete identities that exist within a metadata slot

`getMetas` for returning all metadata slots of an `object`

`isMeta` for testing whether something is the name of a metadata slot

Examples

```
example(importDittoBulk, echo = FALSE)
meta("groups", object = myRNA)
```

`metaLevels` *Gives the distinct values of a meta.data slot (or ident)*

Description

Gives the distinct values of a meta.data slot (or ident)

Usage

```
metaLevels(meta, object, cells.use = NULL, used.only = TRUE)
```

Arguments

`meta`	quoted "meta.data.slot" name = REQUIRED. the meta.data slot whose potential values should be retrieved.
`object`	A target Seurat or SingleCellExperiment object
`cells.use`	String vector of cells'/samples' names which should be included. Alternatively, a Logical vector, the same length as the number of cells in the object, which sets which cells to include. For the typically easier logical method, provide `USE` in `object@cell.names[USE]` OR `colnames(object)[USE]`).
`used.only`	TRUE by default, whether unused levels of already

Value

Returns the distinct values of a metadata slot (factor or not) among to all cells/samples, or for a subset of cells/samples. (Alternatively, returns the distinct values of clustering if `meta = "ident"` and the object is a `Seurat` object).

Author(s)

Daniel Bunis

See Also

`meta` for returning an entire metadata slots of an `object`, not just the potential levels

`getMetas` for returning all metadata slots of an `object`

`isMeta` for testing whether something is the name of a metadata slot

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)

metaLevels("clustering", object = myRNA)

# Note: Set 'used.only' (default = TRUE) to FALSE to show unused levels
```

```
#  of metadata that are already factors.  By default, only the in use options
#  of a metadata are shown.
metaLevels("clustering", myRNA,
    used.only = FALSE)
```

`multi_dittoDimPlot` *Generates multiple dittoDimPlots arranged in a grid.*

Description

Generates multiple dittoDimPlots arranged in a grid.

Usage

```
multi_dittoDimPlot(
  object,
  vars,
  legend.show = FALSE,
  ncol = NULL,
  nrow = NULL,
  axes.labels.show = FALSE,
  xlab = NA,
  ylab = NA,
  OUT.List = FALSE,
  ...
)
```

Arguments

`object`	A Seurat or SingleCellExperiment object to work with
`vars`	c("var1","var2","var3",...). A list of vars from which to generate the separate plots
`legend.show, xlab, ylab, ...`	other paramters passed to `dittoDimPlot`.
`ncol, nrow`	Integer/NULL. How many columns or rows the plots should be arranged into
`axes.labels.show`	Logical. Whether a axis labels should be shown. Ignored if xlab or ylab are set manually.
`OUT.List`	Logical. (Default = FALSE) When set to `TRUE`, a list of the individual plots, named by the `vars` being shown in each, is output instead of the combined multi-plot.

Value

Given multiple 'var' parameters to `vars`, this function will output a dittoDimPlot for each one, arranged into a grid, with some slight tweaks to the defaults. If `OUT.list` was set to TRUE, the list of individual plots, named by the `vars` being shown in each, is output instead of the combined multi-plot. All parameters that can be adjusted in dittoDimPlot can be adjusted here, but the only parameter that can be adjusted between each is the `var`.

Author(s)

Daniel Bunis

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

genes <- getGenes(myRNA)[1:5]
multi_dittoDimPlot(myRNA, c(genes, "clustering"))
```

`multi_dittoDimPlotVaryCells` *Generates multiple dittoDimPlots, each showing different cells, arranged into a grid.*

Description

Generates multiple dittoDimPlots, each showing different cells, arranged into a grid.

Usage

```
multi_dittoDimPlotVaryCells(
  object,
  var,
  vary.cells.meta,
  vary.cells.levels = metaLevels(vary.cells.meta, object),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  min = NULL,
  max = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  show.titles = TRUE,
  show.allcells.plot = TRUE,
  allcells.main = "All Cells",
  show.legend.single = TRUE,
  show.legend.plots = FALSE,
  show.legend.allcells.plot = FALSE,
  nrow = NULL,
  ncol = NULL,
  OUT.List = FALSE,
  ...
)
```

Arguments

object	A Seurat or SingleCellExperiment object to work with
var	String name of a "gene" or "metadata" (or "ident" for a Seurat object) to use for coloring the plots. This is the data that will be displayed for each cell/sample. Alternatively, can be a vector of same length as there are cells/samples in the object. Discrete or continuous data both work. REQUIRED.
vary.cells.meta	String name of a metadata that should be used for selecting which cells to show in each "varycells" plot. REQUIRED.
vary.cells.levels	The values/groupings of the vary.cells.meta metadata that should get a plot. Defaults to all levels of the metadata.
color.panel, colors, min, max, assay, slot, adjustment, ...	additional parameters passed to dittoDimPlot. All parameters except for cells.use, main, and legend.show can be used. A few suggestions: reduction.use for setting which dimensionality reduction space to use. xlab and ylab can be set to NULL to remove the axes labels and provide extra room for the data. size can be used to adjust the size of the dots.
show.titles	Logical which sets whether titles should be added to the individual varycells plots
show.allcells.plot	Logical which sets whether an additional plot showing all of the cells should be added.
allcells.main	String which adjusts the title of the allcells plot. Default = "All Cells". Set to NULL or "" to remove.
show.legend.single	Logical which sets whether to add a single legend as an additional plot. Default = TRUE.
show.legend.plots	Logical which sets whether or not legends should be plotted in varycells plot. Default = FALSE.
show.legend.allcells.plot	Logical which sets whether or a legend should be plotted in the allcells plot. Default = FALSE.
ncol, nrow	Integers which set dimensions of the plot grid.
OUT.List	Logical which controls whether the list of plots should be returned as a list instead of as a single grid arrangement of the plots.

Details

This function generates separate dittoDimPlots that show the same target data, but for distinct cells. Which cells fall into which plot is controlled with the vary.cells.meta parameter. When the quoted name of a metadata containing discrete groupings is given to vary.cells.meta, the function makes separate plots containing all cells/samples of each grouping.

If plots for only certain groupings of cells are wanted, names of the wanted groupings can be supplied to the vary.cells.levels input.

The function then appends a plot containing all groupings, titled as "All Cells" (unless otherwise changed with the allcells.main parameter), as well as a single legend. Either of these can be turned off with the show.allcells.plot and show.legend.single parameters.

Plots are either output in a grid (default) with `ncol` columns and `nrow` rows, or alternatively as a simple list of ggplots if `OUT.List` is set to `TRUE`. In the list, the varycells plots will be named by the value of `vary.cells.meta` that they contain, the allcells plot will be named "allcells" and the single legend will be named "legend".

Either continuous or discrete `var` data can be displayed.

- For continuous data, the range of potential values is calculated at the start, and set, so that colors represent the same values accross all plots.
- For discrete data, colors used in each plot are adjusted so that colors represent the same groupings accross all plots.

Value

multiple dittoDimPlot `ggplot`s either arranged in a grid OR as a list

Author(s)

Daniel Bunis

See Also

`dittoDimPlot` for the base DimPlot plotting function

`multi_dittoDimPlot` for plotting distinct `var`s accross plots instead of disctinct cells

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

multi_dittoDimPlotVaryCells(myRNA, "gene1", vary.cells.meta = "clustering")

# This function can be used to quickly scan for differences in expression
#   within or accross clusters/cell types by providing a gene to 'var'
multi_dittoDimPlotVaryCells(myRNA, "gene1", vary.cells.meta = "clustering")

# This function is also great for generating separate plots of each individual
#   element of a tsne/PCplot/umap. This can be useful to check for dispersion
#   of groups that might otherwise be hidden behind other cells/samples.
#   To do so, set 'var' and 'vary.cells.meta' the same.
multi_dittoDimPlotVaryCells(myRNA, "clustering", vary.cells.meta = "clustering")

# The function can also be used to quickly visualize how separate clustering
#   resolutions match up to each other, or perhaps how certain conditions of
#   cells disperse accross clusters.
multi_dittoDimPlotVaryCells(myRNA, "groups", vary.cells.meta = "clustering")


# For an alternative method of viewing, and easily quantifying, how discrete
#   conditions of cells disperse accross clusters, see '?dittoBarPlot'
```

```
# Note, for displaying expression or scoring of distinct genes or metadata,
#   use 'multi_dittoDimPlot'.  Its split.by variable can then be used to add
#   a varyCells-like effect.
```

`multi_dittoPlot` *Generates multiple dittoPlots arranged into a grid.*

Description

Generates multiple dittoPlots arranged into a grid.

Usage

```
multi_dittoPlot(
  object,
  vars,
  group.by,
  color.by = group.by,
  legend.show = FALSE,
  ncol = 3,
  nrow = NULL,
  main = "var",
  ylab = NULL,
  xlab = NULL,
  OUT.List = FALSE,
  ...
)
```

Arguments

`object`	the Seurat or SingleCellExperiment object to draw from
`vars`	c("var1","var2","var3",...). A vector of gene or metadata names from which to generate the separate plots
`group.by`	String representing the name of a metadata to use for separating the cells/samples into discrete groups.
`color.by`	String representing the name of a metadata to use for setting color. Default = `group.by`.
`ncol, nrow`	Integers which set how many plots will be arranged per column or per row. Default = 3 columns aand however many rows are required. Set both to NULL to have the grid.arrange function figure out what might be most "square" on its own.
`main, ylab`	String which sets whether / how plot titles or y-axis labels should be added to each individual plot • When set to `"var"`, the `vars` names alone will be used. • When set to `"make"`, the default dittoPlot behavior will be observed: Equivalent to "make" for `main`, but for y-axis labels, gene vars will become "'var' expression".

- When set as any other string, that string will be used as the title / y-axis label for every plot.
- When set to NULL, titles / axes labels will not be added.

`xlab, legend.show, ...`
: other paramters passed along to dittoPlot.

`OUT.List`
: Logical. (Default = FALSE) When set to TRUE, a list of the individual plots, named by the `vars` being shown in each, is output instead of the combined multi-plot.

Value

Given multiple 'var' parameters, this function will output a dittoPlot for each one, arranged into a grid, just with some slight tweaks to the defaults. If `OUT.list` was set to TRUE, the list of individual plots is output instead of the combined multi-plot. All parameters that can be adjusted in dittoPlot can be adjusted here.

Author(s)

Daniel Bunis

See Also

dittoPlot for the single plot version of this function

Examples

```
# dittoSeq handles bulk and single-cell data quit similarly.
# The SingleCellExperiment object structure is used for both,
# but all functions can be used similarly directly on Seurat
# objects as well.

example(importDittoBulk, echo = FALSE)
myRNA

genes <- getGenes(myRNA)[1:4]
multi_dittoPlot(myRNA, genes, group.by = "clustering")

# violin-plots in front is often better for large single-cell datasets,
# but we cn change the order with 'plots'
multi_dittoPlot(myRNA, genes, "clustering",
    plots = c("vlnplot","boxplot","jitter"))

#To make it output a grid that is 2x2, to add y-axis labels
# instead of titles, and to show legends...
multi_dittoPlot(myRNA, genes, "clustering",
    nrow = 2, ncol = 2,           #Make grid 2x2 (only one of these needed)
    main = NULL, ylab = "make",   #Add y axis labels instead of titles
    legend.show = TRUE)           #Show legends

# We can also facet with 'split.by'
multi_dittoPlot(myRNA, genes, "clustering",
    split.by = "SNP")
```

setBulk *Set whether a SingleCellExperiment object should be treated as single-cell versus bulk*

Description

Set whether a SingleCellExperiment object should be treated as single-cell versus bulk

Usage

```
setBulk(object, set = TRUE)

## S4 method for signature 'SingleCellExperiment'
setBulk(object, set = TRUE)
```

Arguments

object	A target SingleCellExperiment object
set	Logical, whether the object should be considered as bulk (TRUE) or not (FALSE)

Value

A SingleCellExperiment object with "bulk" internal metadata set to set

Examples

```
example(importDittoBulk, echo = FALSE)
myRNA

isBulk(myRNA)

scRNA <- setBulk(myRNA, FALSE)
isBulk(scRNA)

# Now, if we make a heatmap with this data, we will see that single-cell
# defaults (ordering by the first 'annot.by' & cell names not shown) are used.
dittoHeatmap(scRNA, getGenes(scRNA)[1:30],
    annot.by = c("clustering", "groups"),
    main = "isBulk(object) == FALSE")
```

Simulate *Simulates what a colorblind person would see for any dittoSeq plot!*

Description

Essentially a wrapper function for colorspace's deutan(), protan(), and tritan() functions. This function will output any dittoSeq plot as it might look to an individual with one of the common forms of colorblindness: deutanopia/deutanomaly, the most common, is when the cones mainly responsible for red vision are defective. Protanopia/protanomaly is when the cones mainly responsible for green vision are defective. In tritanopia/tritanomaly, the defective cones are responsible for blue vision. Note: there are more severe color deficiencies that are even more rare. Unfortunately, for these types of color vision deficiency, only non-color methods, like lettering or shapes, will do much to help.

Usage

```
Simulate(
  type = c("deutan", "protan", "tritan"),
  plot.function,
  ...,
  color.panel = dittoColors(),
  min.color = "#F0E442",
  max.color = "#0072B2"
)
```

Arguments

`type`	The type of colorblindness that you want to simulate for. Options: "deutan", "protan", "tritan". Anything else, and you will get an error.
`plot.function`	The plotting function that you want to use/simulate. not quoted. and make sure to remove the () that R will try to add.
`...`	other paramters that can be given to dittoSeq plotting functions, including color.panel, used in exactly the same way they are used for those functions. (contrary to the look of this documentation, color.panel will still default to dittoColors() when not provided.)
`color.panel, min.color, max.color`	The set of colors to be used.

Value

Outputs a dittoSeq plot with the color.panel / min.color & max.color updated as it might look to a colorblind individual.

Note: Does not currently adjust dittoHeatmap.

Author(s)

Daniel Bunis

Examples

```
example(importDittoBulk, echo = FALSE)
Simulate("deutan", dittoDimPlot, object=myRNA, var="clustering", size = 2)
Simulate("protan", dittoDimPlot, myRNA, "clustering", size = 2)
Simulate("tritan", dittoDimPlot, myRNA, "clustering", size = 2)
```

Index

∗ **datasets**
demuxlet.example, 9

addDimReduction, 2, *4*, *20*
addPrcomp, *3*, 4

Darken, 5, *42*
demux.calls.summary, 6, *8*, *49*
demux.SNP.summary, *7*, 7, *49*
demuxlet.example, 9
DGEList, *51*
dittoBarPlot, 9, *20*, *42*
dittoBoxPlot, *30*
dittoBoxPlot *(*dittoPlot*)*, 25
dittoColors, 13, *16*, *39*
dittoDimPlot, *3, 4*, 14, *41, 42*, *46*, *59*, *61, 62*
dittoHeatmap, 21, *42*
dittoPlot, *8*, *20*, 25, *36*, *42*, *64*
dittoPlotVarsAcrossGroups, *30*, 31
dittoRidgeJitter, *30*
dittoRidgeJitter *(*dittoPlot*)*, 25
dittoRidgePlot, *30*
dittoRidgePlot *(*dittoPlot*)*, 25
dittoScatterPlot, *20*, 37
dittoSeq, 42

gene, *16*, *22*, *27*, *33*, *39*, *42*, 43, *44*, *54*
geom_violin, *28*, *34*
getGenes, *19*, *41*, 44, *54*
getMetas, *19*, *41, 42*, 45, *55*, *57, 58*
getReductions, 46
ggplot, *62*
ggrepel, *17*

importDemux, *6–8*, 47
importDittoBulk, *3, 4*, *20*, 50
importDittoBulk,DGEList-method
*(*importDittoBulk*)*, 50
importDittoBulk,list-method
*(*importDittoBulk*)*, 50
importDittoBulk,SummarizedExperiment-method
*(*importDittoBulk*)*, 50
isBulk, 53
isGene, *44*, 54
isMeta, *42*, *45*, 55, *57, 58*

Lighten, *42*, 56

max, *33*
mean, *33*
median, *33*
meta, *42*, *45*, *55*, 57, *58*
metaLevels, *24*, *57*, 58
multi_dittoDimPlot, 59, *62*
multi_dittoDimPlotVaryCells, 60
multi_dittoPlot, *30*, *36*, 63

pbmc_small, *9*
pheatmap, *21–24*

setBulk, 65
setBulk,SingleCellExperiment-method
*(*setBulk*)*, 65
Simulate, *42*, 65
SingleCellExperiment, *3, 4*, *20*, *51, 52*, *65*
slingshot, *18*
SummarizedExperiment, *51*