Package ‘dittoSeq’

April 23, 2024

Type Package

Title User Friendly Single-Cell and Bulk RNA Sequencing Visualization

Version 1.15.8

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 7.3.1

Depends ggplot2

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

Suggests plotly, testthat, Seurat (>= 2.2), DESeq2, edgeR, ggplot.multistats, knitr, rmarkdown, BiocStyle, scRNAseq, ggrastr (>= 0.2.0), ComplexHeatmap, bluster, scater, scran

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

VignetteBuilder knitr

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

git_branch devel

git_last_commit 868cc27

git_last_commit_date 2024-04-01

Repository Bioconductor 3.19

Date/Publication 2024-04-23

Author Daniel Bunis [aut, cre],
Jared Andrews [aut, ctb]

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

Contents

addDimReduction . . . 3
addPrcomp . . . 4
Darken . . . 5
demux.calls.summary . . . 6
demux.SNP.summary . . . 7
demuxlet.example . . . 9
dittoBarPlot . . . 10
dittoColors . . . 14
dittoDimPlot . . . 15
dittoDotPlot . . . 25
dittoFreqPlot . . . 33
dittoHeatmap . . . 41
dittoHex . . . 46
dittoPlot . . . 56
dittoPlotVarsAcrossGroups . . . 66
dittoScatterPlot . . . 73
dittoSeq . . . 81
gene . . . 82
GeneTargeting . . . 83
getGenes . . . 87
getMetas . . . 88
getReductions . . . 89
importDemux . . . 90
importDittoBulk . . . 94
isBulk . . . 97
isGene . . . 98
isMeta . . . 99
Lighten . . . 101
meta . . . 101
metaLevels . . . 103
multi_dittoDimPlot . . . 104
multi_dittoDimPlotVaryCells . . . 106
multi_dittoPlot . . . 109
setBulk . . . 111
Simulate . . . 112

Index 114

`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 Seurat's pbmc_small example data 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"),
  split.by = NULL,
  cells.use = NULL,
  retain.factor.levels = FALSE,
  data.out = FALSE,
  do.hover = FALSE,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  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, SingleCellExperiment, or SummarizedExperiment 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 counts versus percentage.

`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`

`cells.use` String vector of cells'/samples' names OR an integer vector specifying the indices of cells/samples 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.

Note: When `cells.use` is combined with `scale = "percent"`, left out cells are not considered in calculating percentages. Percents will always total to 1.

`retain.factor.levels` Logical which controls whether factor identities of `var` and `group.by` data should be respected. Set to TRUE to faithfully reflect ordering of groupings encoded in factor levels, but Note that this will also force retention of groupings that could otherwise be removed via `cells.use`.

`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.

`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`.)

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

split.adjust
: A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. ‘list(scales = "free")‘.

  For options, when giving 1 metadata to split.by, see facet_wrap, OR when giving 2 metadatas to split.by, see facet_grid.

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.

  Hint: use metaLevels or unique(<var-data>) to assess current values.

var.labels.reorder
: Integer vector. A sequence of numbers, from 1 to the number of distinct var value identities, 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 replace 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.

  Recommendation for advanced users: If you find yourself coming back to this input too many times, an alternative solution that can be easier long-term is to make the target data into a factor, and to put its levels in the desired order: factor(data, levels = c("level1", "level2", ...)). metaLevels can be used to quickly get the identities that need to be part of this 'levels' input.

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. Default = "make" and if left as make, a title will be automatically generated.

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

See Also

`dittoFreqPlot` for a data representation that focuses on pre-sample frequencies of each the `var`-data values individually, rather than emphasizing total makeup of samples/groups.

Examples

```
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)
    }

### Previous Version Compatibility
# Mistakenly, dittoBarPlot used to remove factor identities entirely from the
#  data it used. This manifests as ignorance of a user's set orderings for
#  their data. That is nolonger done by default, but to recreate old plots,
#  restoring this behavior can be achieved with 'retain.factor.levels = FALSE'
# Set factor level ordering for a metadata we'll give to 'group.by'
myRNA$groups_reverse_levels <- factor(
    myRNA$groups,
    levels = c("D", "C", "B", "A"))
# dittoBarPlot will now respect this level order by default.
dittoBarPlot(myRNA, "clustering", group.by = "groups_reverse_levels")
# But that respect can be turned off...
dittoBarPlot(myRNA, "clustering", group.by = "groups_reverse_levels",
    retain.factor.levels = FALSE)
```

`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,
  split.adjust = list(),
  extra.vars = NULL,
  multivar.split.dir = c("col", "row"),
  show.others = TRUE,
  split.show.all.others = TRUE,
  split.nrow = NULL,
  split.ncol = NULL,
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  swap.rownames = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  shape.panel = c(16, 15, 17, 23, 25, 8),
  min.color = "#F0E442",
  max.color = "#0072B2",
  min = NA,
  max = NA,
  order = c("unordered", "increasing", "decreasing", "randomize"),
  main = "make",
  sub = NULL,
  xlab = "make",
  ylab = "make",
  rename.var.groups = NULL,
  rename.shape.groups = NULL,
  theme = theme_bw(),
  show.axes.numbers = TRUE,
  show.grid.lines = if (is.character(reduction.use)) {
     !grepl("umap|tsne",
    tolower(reduction.use))
 } else {
     TRUE
 },
  do.letter = FALSE,
  do.ellipse = FALSE,
  do.label = FALSE,
```

```
  labels.size = 5,
  labels.highlight = TRUE,
  labels.repel = TRUE,
  labels.split.by = split.by,
  labels.repel.adjust = list(),
  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,
  do.contour = FALSE,
  contour.color = "black",
  contour.linetype = 1,
  legend.show = TRUE,
  legend.size = 5,
  legend.title = "make",
  legend.breaks = waiver(),
  legend.breaks.labels = waiver(),
  shape.legend.size = 5,
  shape.legend.title = shape.by,
  do.raster = FALSE,
  raster.dpi = 300,
  data.out = FALSE
)
```

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment object.

`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, a string vector naming multiple genes or metadata, OR a vector of the same length as there are cells/samples in the `object` which provides per-cell data directly.

`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 with "umap", "tsne", or "pca" within its name, (priority: UMAP > t-SNE > PCA) or the first dimensionality reduction slot if none of those exist.

Alternatively, a matrix (or data.frame) containing the dimensionality reduction embeddings themselves. The matrix should have as many rows as there are cells/samples in the `object`. Note that `dim.1` and `dim.2` will still be used to

select which columns to pull from, and column names will serve as the default `xlab` & `ylab`.

`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 OR an integer vector specifying the indices of cells/samples 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.

`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`

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. 'list(scales = "free")'.

For options, when giving 1 metadata to `split.by`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

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

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

`multivar.split.dir`
"row" or "col", sets the direction of faceting used for 'var' values when `var` is given multiple genes or metadata, and when `split.by` is used to provide additional data to facet by.

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

`split.show.all.others`
Logical which sets whether gray "others" cells of facets should include all cells of other facets (`TRUE`) versus just cells left out by `cell.use` (`FALSE`).

`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 integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`adjustment`
: When plotting gene / feature expression, 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]

`swap.rownames`
: optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

`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.

`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.

`order`
: String. If the data should be plotted based on the order of the color data, sets whether to plot (from back to front) in "increasing", "decreasing", "randomize" order. If left as "unordered", plot order is simply based on the order of cells within the `object`.

`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`.

`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.

`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.

`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 = FALSE for umap and tsne `reduction.use`, TRUE otherwise.

`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.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.

`labels.split.by` String of one or two metadata names which controls the facet-split calculations for label placements. Defaults to `split.by`, so generally there is no need to adjust this except when you are utilizing the `extra.vars` input to achieve manual faceting control.

`labels.repel.adjust` A named list which allows extra parameters to be pushed through to ggrepel function calls. List elements should be valid inputs to the `geom_label_repel` by default, or `geom_text_repel` when `labels.highlight = FALSE`.

`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 provide `add.trajectory.lineages = slingLineages('object')`.

`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 = slingCurves('object')`

`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.

`do.contour`
: Logical. Whether density-based contours should be displayed.

`contour.color`
: String that sets the color(s) of the `do.contour` contours.

`contour.linetype`
: String or numeric which sets the type of line used for `do.contour` contours. Defaults to "solid", but see `linetype` for other options.

`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.

`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.

`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.raster`
: Logical. When set to `TRUE`, rasterizes the internal plot area. Useful for editing in external programs (e.g. Illustrator).

`raster.dpi`
: Number indicating dpi to use for rasterization. Default = 300.

`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").

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 do.contour is provided, density gradiant contour lines will be overlaid with color and linetype adjustable via contour.color and contour.linetype.
- 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 and Jared Andrews

See Also

getGenes and getMetas to see what the var, split.by, etc. options are of an object.

getReductions to see what the reduction.use options are of an object.

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.

dittoDimHex and dittoScatterHex for showing very similar data representations, but where nearby cells are summarized together in hexagonal bins.

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

dittoBarPlot for an alternative discrete data display and quantification method.

Examples

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

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

# You can also plot multiple sets of continuous data:
dittoDimPlot(myRNA, c("gene1", "gene2"))
# (See ?multi_dittoDimPlot if you would like to have wholy separate
# plots/scales/legends for each set.)

# 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")

dittoDimPlot(myRNA, "gene1",
    do.contour = TRUE,
    contour.color = "lightblue", # Optional, black by default
    contour.linetype = "dashed") # Optional, solid by default

# Plotting ordering can also be adjusted with 'order':
dittoDimPlot(myRNA, "timepoint", size = 20,
    order = "increasing")
dittoDimPlot(myRNA, "timepoint", size = 20,
    order = "decreasing")
dittoDimPlot(myRNA, "timepoint", size = 20,
    order = "randomize")
```

`dittoDotPlot` *Compact plotting of per group summaries for expression of multiple features*

Description

Compact plotting of per group summaries for expression of multiple features

Usage

```
dittoDotPlot(
  object,
  vars,
  group.by,
  scale = TRUE,
  split.by = NULL,
  cells.use = NULL,
  size = 6,
  vars.dir = c("x", "y"),
  categories.split.adjust = TRUE,
  categories.theme.adjust = TRUE,
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  min.color = "grey90",
  max.color = "#C51B7D",
  min = "make",
  max = NA,
  mid.color = NULL,
  mid = "make",
  summary.fxn.color = function(x) {
     mean(x[x != 0])
 },
```

```
  summary.fxn.size = function(x) {
     mean(x != 0)
 },
  min.percent = 0.01,
  max.percent = NA,
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  swap.rownames = NULL,
  do.hover = FALSE,
  main = NULL,
  sub = NULL,
  ylab = group.by,
  y.labels = NULL,
  y.reorder = NULL,
  xlab = NULL,
  x.labels.rotate = vars.dir == "x",
  groupings.drop.unused = TRUE,
  theme = theme_classic(),
  legend.show = TRUE,
  legend.color.breaks = waiver(),
  legend.color.breaks.labels = waiver(),
  legend.color.title = "make",
  legend.size.title = "percent\nexpression",
  data.out = FALSE
)
```

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment object.

`vars` String vector of gene or metadata names which selects the features to summarize and show. Example: `c("gene1","gene2","gene3")`

Alternatively, a named list of string vectors where names represent category labels, such as associated cell types, and values are the gene or metadata names that you wish to have grouped together. Example: `vars = list('Epithelial Cells' = c("gene1","gene2"), Neuron = c("gene3"))`

`group.by` String representing the name of a metadata to use for separating the cells/samples into discrete groups.

`scale` String which sets whether the values shown with color (default: mean non-zero expression) should be centered and scaled.

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

- When 2 metadata 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`

- Note: When `vars` are provided in list format, to group its contents into categories, that grouping is carried out via faceting and takes up one of the `split.by` slots.

`cells.use` String vector of cells'/samples' names OR an integer vector specifying the indices of cells/samples 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.

`size` Number which sets the visual dot size associated with the highest value shown by dot size (default: percent non-zero expression).

`vars.dir` "x" or "y", sets the axis where `vars` will be displayed.

`categories.split.adjust`
Boolean. When `TRUE` (default), and `vars`-categories have been provided, improves category display by:

- adding `list(switch = "y", scales = "free_y", space = "free_y")` to the default for `split.adjust` (or 'x' counterparts depending on `vars.dir`)
- enforcing that `facet_grid` will be used for faceting because `facet_wrap` cannot receive the 'space' argument.

`categories.theme.adjust`
Boolean. When `TRUE` (default), and `vars`-categories have been provided, improves category display by adding `theme(strip.placement = "outside", strip.background.y = element_blank())` to the given `theme` (or 'x' counterpart depending on `vars.dir`)

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

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. 'list(scales = "free")'.

For options, when giving 1 metadata to `split.by`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

`min.color`, `max.color`
colors to use for minimum and maximum color values. Default = light grey and purple. Ignored if `mid.color` given as `"ryb"`, `"rwb"`, or `"rgb"` which will update these to be "blue" and "red", respectively.

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

`mid.color` NULL (default), "ryb", "rwb", "rgb", or a color to use for the midpoint of a three-color color scale. *This parameter acts a switch between using a 2-color scale or a 3-color scale*:

- When left NULL, the 2-color scale runs from `min.color` to `max.color`, using `scale_fill_gradient`.
- When given a color, the 3-color scale runs from `min.color` to `mid.color` to `max.color`, using `scale_fill_gradient2`.
- When given `"ryb"`, `"rwb"`, *or* `"rgb"` *serves as a* **single-point, quick switch to a "standard" 3-color scale** by also updating the `min.color` and `max.color`. Doing so sets:

- `max.color` to a red,
- `min.color` to a blue,
- and `mid.color` to either a yellow ("r*y*b"), "white" ("r*w*b"), or "gray97" ("r*g*b", gray not green).
- Actual colors used are inspired by ColorBrewer "RdYlBu" and "RdBu" palettes.

Thus, the 3-color scale runs from a blue to one of a yellow, "white", or "gray97" to a red, using `scale_fill_gradient2`.

`mid` Number or "make" (default) which sets the value associated with the `mid.color` of the three-color scale. Ignored when `mid.color` is left as NULL. When "make", defaults to midway between what dittoSeq expects to be the minimum and maximum values shown in the legend. (Maps to the 'midpoint' parameter of `scale_fill_gradient2`.)

`summary.fxn.color`, `summary.fxn.size`
A function which sets how color or size will be used to summarize variables' data for each group. Any function can be used as long as it takes in a numeric vector and returns a single numeric value.

`min.percent`, `max.percent`
Numbers between 0 and 1 which sets the minimum and maximum percent expression to show. When set to NA, the minimum/maximum of the data are used.

`assay`, `slot` single strings or integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`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]

`swap.rownames` optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

`do.hover` Logical. Default = `FALSE`. If set to `TRUE` the object will be converted to an interactive plotly object in which underlying data for individual dots will be displayed when you hover your cursor over them.

`main` String which sets the plot title.

`sub` String which sets the plot subtitle.

`ylab` String which sets the y/grouping-axis label. Default is `group.by` so it defaults to the name of the grouping information. Set to `NULL` to remove.

`y.labels` String vector, c("label1","label2","label3",...) which overrides the names of the samples/groups.

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

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

Recommendation for advanced users: If you find yourself coming back to this input too many times, an alternative solution that can be easier long-term is to make the target data into a factor, and to put its levels in the desired order: `factor(data, levels = c("level1", "level2", ...))`. `metaLevels` can be used to quickly get the identities that need to be part of this 'levels' input.

`xlab` String which sets the x/var-axis label. Set to `NULL` to remove.

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

`groupings.drop.unused`
Logical. `TRUE` by default. If `group.by`-data is a factor, factor levels are retained for ordering purposes, but some level(s) can end up with zero cells left after `cells.use` subsetting. By default, we remove them, but you can set this input to `FALSE` to keep them.

`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.

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

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

`legend.color.breaks.labels`
String vector, with same length as `legend.breaks`, which sets the labels for the tick marks of the color-scale.

`legend.color.title`, `legend.size.title`
String or `NULL`, sets the title displayed above legend keys.

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

Details

This function will output a compact summary of expression of multiple genes, or of values of multiple numeric metadata, across cell/sample groups (clusters, sample identity, conditions, etc.), where dot-size and dot-color are used to reflect distinct features of the data. Typically, and by default, size will reflect the percent of non-zero values, and color will reflect the mean of non-zero values for each var and group pairing.

Internally, 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. (Note that 'adjustment' would be applied *before* cells/samples subsetting, and across all groups of cells/samples.)

Groupings are determined using `group.by`, and then data for each variable is summarized based on `summary.fxn.color` & `summary.fxn.size`.

If `scale = TRUE` (default setting), the color summary values are centered and scaled. Doing so 1) puts values for all `vars` in a similar range, and 2) emphasizes relative differences between groups.

Finally, data is plotted as dots of differing colors and sizes, with `vars` along the `vars.dir`-axis and groupings along the other. Labels along the x-axis can be rotated 45 degrees with `x.label.rotate=TRUE`, which is on by default when `vars.dir=='x'`.

Value

a ggplot object where dots of different colors and sizes summarize continuous data for multiple features per multiple groups.

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 the dots.

Many characteristics of the plot can be adjusted using discrete inputs

- Size of the dots can be changed with `size`.
- Subsetting to utilize only certain cells/samples can be achieved with `cells.use`.
- Markers can be grouped into categories by providing them to the `vars` input as a list, where list element names represent category names, and list element contents are the feature names which each category should contain.
- Colors (2-color scale) can be adjusted with `min.color` and `max.color`.
- Coloring can also be switched to a 3-color scale by using the `mid.color` parameter. For details, see that parameter's description above.
- Displayed value ranges can be adjusted with `min` and `max` for color, or `min.percent` and `max.percent` for size.
- Titles and axes labels can be adjusted with `main`, `sub`, `xlab`, `ylab`, `legend.color.title`, and `legend.size.title` arguments.
- The legend can be hidden by setting `legend.show = FALSE`.
- The color legend tick marks and associated labels can be adjusted with `legend.color.breaks` and `legend.color.breaks.labels`, respectively.
- The groupings labels and order can be changed using `y.labels` and `y.reorder`
- Rotation of x-axis labels can be turned off with `x.labels.rotate = FALSE`.

Author(s)

Daniel Bunis

See Also

`dittoPlotVarsAcrossGroups` for a different method of summarizing expression of multiple features across distinct groups that can be better (and more compact) when the mapping of values to individual genes among the requested set are unimportant.

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

Examples

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

# These random data don't mimic dropout, so we'll add some zeros.
logcounts(myRNA)[
    matrix(
        sample(c(TRUE,FALSE), ncol(myRNA)*10, p=c(.2,.8), replace = TRUE),
        ncol=10
    )] <- 0

dittoDotPlot(
    myRNA, c("gene1", "gene2", "gene3", "gene4"),
    group.by = "clustering")

# 'size' adjusts the dot-size associated with the highest percent expression
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    size = 12)

# 'scale' input can be used to control / turn off scaling of avg exp values.
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    scale = FALSE)

# x-axis label rotation can be controlled with 'x.labels.rotate'
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    x.labels.rotate = FALSE)

# The axis that vars get shown on can be swapped with the 'vars.dir' input.
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    vars.dir = "y")

# Titles are adjustable via various discrete inputs:
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    main = "Title",
    sub = "Subtitle",
    ylab = "y-axis label",
    xlab = "x-axis label",
    legend.color.title = "Colors title",
    legend.size.title = "Dot size title")

# You can also bin vars into groups by providing them in a named list:
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(
        'Naive' = c("gene1", "gene2"),
        'Stimulated' = c("gene3", "gene4")
    )
)
# The 'categories.split.adjust' and 'categories.theme.adjust' arguments then
#   control whether 'split.adjust' and 'theme' input contents, respectively,
#   will be added to in ways that make these categories actually appear, and
#   work, like categories.
# They both default to TRUE, and the axis they affect follows 'vars.dir'.
```

```
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(Naive = c("gene1", "gene2"), Stimulated = c("gene3"))
)
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(Naive = c("gene1", "gene2"), Stimulated = c("gene3")),
    split.by = "conditions"
)
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(Naive = c("gene1", "gene2"), Stimulated = c("gene3")),
    categories.split.adjust = FALSE,
    categories.theme.adjust = FALSE
)
# Now with 'vars.dir' changed to 'y'...
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(Naive = c("gene1", "gene2"), Stimulated = c("gene3")),
    vars.dir = "y"
)
dittoDotPlot(myRNA, group.by = "clustering",
    vars = list(Naive = c("gene1", "gene2"), Stimulated = c("gene3")),
    split.by = "conditions",
    vars.dir = "y"
)

# Coloring can be swapped from the default 2-color scale to a 3-color scale
#   by using the 'mid.color' input:
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    mid.color = "white"
)
# Setting it to "ryb", "rgb", or "rwb" quickly updates this input as well as
#   'min.color' and 'max.color', making the affect of these next two calls
#   equivalent:
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    mid.color = "rgb"
)
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    min.color = "#2166AC", # (blue)
    mid.color = "gray97",  # (gray)
    max.color = "#B2182B"  # (red)
)

# For certain specialized applications, it may be helpful to adjust the
#   functions used for summarizing the data as well. Inputs are:
#   summary.fxn.color & summary.fxn.size
#     Requirement for each: Any function that takes in a numeric vector &
#     returns, as output, a single numeric value.
dittoDotPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    summary.fxn.color = mean,
    legend.color.title = "mean\nexpression\nincluding 0s",
    x.labels.rotate = FALSE,
    scale = FALSE)
```

dittoFreqPlot *Plot cell type/cluster/identity frequencies per sample and per grouping*

Description

Plot cell type/cluster/identity frequencies per sample and per grouping

Usage

```
dittoFreqPlot(
  object,
  var,
  sample.by = NULL,
  group.by,
  color.by = group.by,
  vars.use = NULL,
  scale = c("percent", "count"),
  max.normalize = FALSE,
  plots = c("boxplot", "jitter"),
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  cells.use = NULL,
  data.out = FALSE,
  do.hover = FALSE,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  y.breaks = NULL,
  min = 0,
  max = NA,
  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,
  jitter.size = 1,
  jitter.width = 0.2,
  jitter.color = "black",
  jitter.position.dodge = boxplot.position.dodge,
  do.raster = FALSE,
  raster.dpi = 300,
  boxplot.width = 0.4,
```

```
  boxplot.color = "black",
  boxplot.show.outliers = NA,
  boxplot.outlier.size = 1.5,
  boxplot.fill = TRUE,
  boxplot.position.dodge = vlnplot.width,
  boxplot.lineweight = 1,
  vlnplot.lineweight = 1,
  vlnplot.width = 1,
  vlnplot.scaling = "area",
  vlnplot.quantiles = NULL,
  ridgeplot.lineweight = 1,
  ridgeplot.scale = 1.25,
  ridgeplot.ymax.expansion = NA,
  ridgeplot.shape = c("smooth", "hist"),
  ridgeplot.bins = 30,
  ridgeplot.binwidth = NULL,
  add.line = NULL,
  line.linetype = "dashed",
  line.color = "black",
  legend.show = TRUE,
  legend.title = color.by
)
```

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment 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`.

`sample.by` String name of a metadata containing which samples each cell belongs to.

Note that when this is not provided, there will only be one data point per grouping. A warning can be expected then for all `plots` options except `"jitter"`, but this can be a useful exercise when simply trying to quantify cell type frequency fluctuations among 2 particular metadata (given to group.by & color.by) combinations.

`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 supersets or subgroups when wanted, but it defaults to `group.by` so this input can be skipped otherwise.

`vars.use` String or string vector naming a subset of the values of `var`-data which should be shown. If left as `NULL`, all values are shown.

Hint: use `metaLevels` or `unique(<var-data>)` to assess options.

Note: When `var.labels.rename` is jointly utilized to update how the `var`-values are shown, the **updated** values must be used.

`scale` "count" or "percent". Sets whether data should be shown as counts versus percentage.

`max.normalize` Logical which sets whether the data for each `var`-data value (each facet) should be normalized to have the same maximum value.

When set to `TRUE`, lower frequency `var`-values will make use of just as much plot space as higher frequency vars.

`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.

`split.nrow`, `split.ncol`

Integers which set the dimensions of the facet grid. (the `split.nrow` and `split.ncol` equivalent of other functions)

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. ‘list(scales = "free")‘.

Faceting for this dittoFreqPlot is always by the `var`-data, so see `facet_wrap` for options.

`cells.use` String vector of cells’(/samples’ for bulk data) names OR an integer vector specifying the indices of cells/samples 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.

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

`do.hover` Logical. Default = `FALSE`. If set to `TRUE`: 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 (assuming that there is a "jitter" in `plots`),

`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`.)

`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 display. Default = NA, which allows ggplot to set these limits based on the range of all data being shown.

`var.labels.rename`

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

Hint: use `metaLevels` or `unique(<var-data>)` to assess current values.

`var.labels.reorder`

Integer vector. A sequence of numbers, from 1 to the number of distinct `var`-value identities, for rearranging the order of facets within the plot space.

Method: Make a first plot without this input. Then, treating the top-left-most grouping as index 1, and the bottom-right-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, c("label1","label2","label3",...) which overrides the names of groupings.

`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.

  Recommendation for advanced users: If you find yourself coming back to this input too many times, an alternative solution that can be easier long-term is to make the target data into a factor, and to put its levels in the desired order: `factor(data, levels = c("level1", "level2", ...))`. `metaLevels` can be used to quickly get the identities that need to be part of this 'levels' input.

`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 grouping-axis label (=x-axis for box and violin plots, y-axis for ridgeplots). Set to `NULL` to remove.

`ylab`
: String, sets the continuous-axis label (=y-axis for box and violin plots, x-axis for ridgeplots). Default = "make" and if left as make, a title will be automatically generated.

`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

`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.

  Note for when `color.by` is used to split x-axis groupings into additional bins: ggplot does not shrink jitter widths accordingly, so be sure to do so yourself! Ideally, needs to be 0.5/num_subgroups.

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

`jitter.position.dodge`
: Scalar which adjusts the relative distance between jitter widths when multiple subgroups exist per `group.by` grouping (a.k.a. when `group.by` and `color.by` are not equal). Similar to `boxplot.position.dodge` input & defaults to the value of that input so that BOTH will actually be adjusted when only, say, `boxplot.position.dodge = 0.3` is given.

`do.raster`
: Logical. When set to `TRUE`, rasterizes the jitter plot layer, changing it from individually encoded points to a flattened set of pixels. This can be useful for editing in external programs (e.g. Illustrator) when there are many thousands of data points.

`raster.dpi`
: Number indicating dots/pixels per inch (dpi) to use for rasterization. Default = 300.

`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.outlier.size`
Scalar which adjusts the size of points used to mark outliers

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

`boxplot.position.dodge`
Scalar which adjusts the relative distance between boxplots when multiple are drawn per grouping (a.k.a. when `group.by` and `color.by` are not equal). By default, this input actually controls the value of `jitter.position.dodge` unless the `jitter` version is provided separately.

`boxplot.lineweight`
Scalar which adjusts the thickness of boxplot lines.

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

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

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

`vlnplot.quantiles`
Single number or numeric vector of values in [0,1] naming quantiles at which to draw a horizontal line within each violin plot. Example: `c(0.1, 0.5, 0.9)`

`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

`ridgeplot.ymax.expansion`
Scalar which adjusts the minimal space between the top-most grouping and the top of the plot in order to ensure that the curve is not cut off by the plotting grid. The larger the value, the greater the space requested. When left as NA, dittoSeq will attempt to determine an ideal value itself based on the number of groups & linear interpolation between these goal posts: 0.6 when g<=3, 0.1 when g==12, and 0.05 when g>=34, where g is the number of groups.

`ridgeplot.shape`
Either "smooth" or "hist", sets whether ridges will be smoothed (the typical, and default) versus rectangular like a histogram. (Note: as of the time shape "hist" was added, combination of jittered points is not supported by the `stat_binline` that dittoSeq relies on.)

`ridgeplot.bins` Integer which sets how many chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Overridden by `ridgeplot.binwidth` when that input is provided.

`ridgeplot.binwidth`
: Integer which sets the width of chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Takes precedence over `ridgeplot.bins` when provided.

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

Details

The function creates a dataframe containing counts and percent makeup of `var` identities per sample if `sample.by` is given, or per group if only `group.by` is given. `color.by` can optionally be used to add subgroupings to calculations and ultimate plots, or to convey super-groups of `group.by` groupings.

Typically, `var` will be pointed to clustering or cell type annotations, but in truth it can be given any discrete data.

If a set of cells to use is indicated with the `cells.use` input, only those cells/samples are used for counts and percent makeup calculations.

If a set of `var`-values to show is indicated with the `vars.use` input, the data.frame is trimmed at the end to include only corresponding rows.

If `max.normalized` is set to `TRUE`, counts and percent data are transformed to a 0-1 scale, which makes better use of white space for lower frequency `var`-values.

Either percent of total (`scale = "percent"`), which is the default, or counts (if `scale = "count"`) data is then (gg)plotted with the data representation types in `plots` by utilizing the same machinery as `dittoPlot`. Faceting by `var`-data values is utilized to achieve per `var`-value (e.g. cluster or cell type) granularity.

See below for additional customization options!

Value

A ggplot plot where frequencies of discrete data, grouped by sample, condition, etc., is shown on 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, 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.

Calculation Details

The function is restricted in that each samples' cells, indicated by the unique values of `sample.by`-data, must exist within single `group.by` and `color.by` groupings. Thus, in order to ensure all valid `var`-data composition data points are generated, prior to calculations...

- var-data are ensured to be a factor, which ensures a calculation will be run for every var-value (a.k.a. cell type or cluster)
- group.by-data and color-by-data are treated as non-factor data, which ensures that calculations are run only for the groupings that each sample is associated with.

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. For example, all jitter adjustments start with "jitter.", such as jitter.size and jitter.width.

Inclusion of "ridgeplot" overrides "boxplot" and "vlnplot" presence and changes the plot to be horizontal.

Additionally:

- **Colors can be adjusted** with color.panel.
- **Subgroupings:** color.by can be utilized to split major group.by groupings into subgroups. When this is done in y-axis plotting, dittoSeq automatically ensures the centers of all geoms will align, but users will need to manually adjust jitter.width to less than 0.5/num_subgroups to avoid overlaps. There are also three inputs through which one can use to control geom-center placement, but the easiest way to do all at once so is to just adjust vlnplot.width! The other two: boxplot.position.dodge, and jitter.position.dodge.
- **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.
- **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 = FALSE.
- **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 on/off with x.labels.rotate = TRUE/FALSE.

Author(s)

Daniel Bunis

See Also

dittoBarPlot for a data representation that emphasizes total makeup of samples/groups rather than focusing on the var-data values individually.

Examples

```
# Establish some workable example data
example(importDittoBulk, echo = FALSE)
myRNA1 <- myRNA
colnames(myRNA) <- paste0(colnames(myRNA),"_1")
example(importDittoBulk, echo = FALSE)
myRNA <- cbind(myRNA, myRNA1)
myRNA <- setBulk(myRNA, FALSE)
myRNA$sample <- rep(1:12, each = 10)
myRNA$groups <- rep(c("A", "B"), each = 60)
myRNA$subgroups <- rep(as.character(c(1:3,1:3,1:3,1:3)), each = 10)
myRNA

# There are three main inputs for this function, in addition to 'object'.
#  var = typically this will be cell types annotations or clustering
#  sample.by = the name of a metadata containing sample assignment of cells.
#  group.by = how to group the data on the x-axis (y-axis for ridgeplots)
dittoFreqPlot(myRNA,
    var = "clustering",
    sample.by = "sample",
    group.by = "groups")

# 'color.by' can also be set differently from 'group.by' to have the effect
#  of highlighting supersets or subgroupings:
dittoFreqPlot(myRNA, "clustering",
    group.by = "groups",
    sample.by = "sample",
    color.by = "subgroups")

# The var-values shown can be subset with 'vars.use'
dittoFreqPlot(myRNA, "clustering",
    group.by = "groups", sample.by = "sample", color.by = "subgroups",
    vars.use = 1:2)

# Lower frequency groups can be expanded to use the entire y-axis by:
#  turning on 'max.normalize'-ation:
dittoFreqPlot(myRNA, "clustering",
    group.by = "groups", sample.by = "sample", color.by = "subgroups",
    max.normalize = TRUE)
#  or by setting y-scale limits to be set by the contents of facets:
dittoFreqPlot(myRNA, "clustering",
    group.by = "groups", sample.by = "sample", color.by = "subgroups",
    split.adjust = list(scales = "free_y"))

# Data representations can also be selected and reordered with the 'plots'
#  input, and further adjusted with inputs applying to each representation.
dittoFreqPlot(myRNA,
    var = "clustering", sample.by = "sample", group.by = "groups",
    plots = c("vlnplot", "boxplot", "jitter"),
    vlnplot.lineweight = 0.2,
    boxplot.fill = FALSE,
    boxplot.lineweight = 0.2)
```

```
# Finally, 'sample.by' is not technically required. When not given, a
#  single-datapoint of overall composition stats will be shown for each
#  grouping.
#  Just note, all data representation other than "jitter" will complain
#  due to there only being the one datapoint per group.
dittoFreqPlot(myRNA,
    var = "clustering", group.by = "groups", color.by = "subgroups",
    plots = "jitter")
```

---

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 either the pheatmap (default) or ComplexHeatmap package.

Usage

```
dittoHeatmap(
  object,
  genes = getGenes(object, assay),
  metas = NULL,
  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),
  swap.rownames = NULL,
  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.features = NULL,
  show_colnames = isBulk(object),
  show_rownames = TRUE,
  scale = "row",
  cluster_cols = isBulk(object),
```

```
  border_color = NA,
  legend_breaks = NA,
  drop_levels = FALSE,
  breaks = NA,
  complex = FALSE,
  ...
)
```

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment object.

`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.

`metas` String vector, c("meta1","meta2","meta3",...) = the list of metadata variables to put in the heatmap.

`cells.use` String vector of cells'/samples' names OR an integer vector specifying the indices of cells/samples 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.

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

`order.by` Single string, string vector, or numeric vector which sets how cells/samples (columns) will be ordered when `cluster_cols = FALSE`.

Strings should be the name of a gene, or metadata slot, but can also be multiple such values in order of priority.

Alternatively, can be a numeric vector which gives the column index order directly.

`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 integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`swap.rownames` optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

`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.features`
String vector of genes/metadata whose names you would like to show. Only these genes/metadata 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.features`, 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`, `drop_levels`, `...`
other arguments passed to `pheatmap` directly (or to `pheatmap` if `complex = TRUE`).

`complex` Logical which sets whether the heatmap should be generated with ComplexHeatmap (`TRUE`) versus pheatmap (`FALSE`, default).

Details

This function serves as a wrapper for creating heatmaps from bulk or single-cell RNAseq data with pheatmap::`pheatmap`, by essentially automating the data extraction and annotation building steps. (Or alternatively with ComplexHeatmap::`pheatmap` if `complex` is set to `true`.

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 `complex` is set to `TRUE`, a `Heatmap`

Alternatively, if `data.out` is 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 features can also be selected for labeling using the `highlight.features` 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
- When `complex` is set to `TRUE`, additional inputs for the `Heatmap` function can be given as well. Some examples:
  - `use_raster` to have the heatmap rasterized/flattened to pixels which can make working with large heatmaps in a figure editor, like Illustrator, simpler.
  - `name` to give the heatmap color scale a custom title.

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 and Jared Andrews

See Also

pheatmap::`pheatmap`, for how to add additional heatmap tweaks, OR or ComplexHeatmap::`pheatmap` and `Heatmap` for when you want to turn on rasterization or any additional customizations offered by this fantastic package.

`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

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

# We now have two SCEs for our example purposes:
  # 'myRNA' will be treated as a bulk RNAseq dataset
  # 'scRNA' will be treated as a single-cell RNAseq dataset

# 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 often 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",
    cluster_cols = FALSE)
# 'order.by' can be multiple metadata/genes, or a vector of indexes directly
dittoHeatmap(scRNA, genes,
    annot.by = "clustering",
    order.by = c("clustering", "timepoint"))
dittoHeatmap(scRNA, genes,
    annot.by = "clustering",
    order.by = ncol(scRNA):1)

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

# When theree are many many cells & genes, rasterization can be super useful
# as well.
#   Rasterization, or flattening of the distinct color objects to a matrix of
#   pixels, is the default for large heatmaps in the ComplexHeatmap package,
#   and you can have the heatmap rendered with this package (rather than the
#   pheatmap package) by setting 'complex = TRUE'.
#   Our data here is too small to hit that defaulting switch, so lets give
#   the direct input, 'use_raster' as well:
if (requireNamespace("ComplexHeatmap")) { # Checks if you have the package.
    dittoHeatmap(scRNA, genes, annot.by = "groups", show_colnames = FALSE,
        complex = TRUE,
        use_raster = TRUE)
}

# 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)
```

dittoHex *Show RNAseq data, grouped into hexagonal bins, on a scatter or dimensionality reduction plot*

Description

Show RNAseq data, grouped into hexagonal bins, on a scatter or dimensionality reduction plot

Usage

```
dittoDimHex(
  object,
  color.var = NULL,
  bins = 30,
  color.method = NULL,
  reduction.use = .default_reduction(object),
  dim.1 = 1,
  dim.2 = 2,
  cells.use = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  split.by = NULL,
  extra.vars = NULL,
  multivar.split.dir = c("col", "row"),
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  swap.rownames = NULL,
  assay.extra = assay,
  slot.extra = slot,
  adjustment.extra = adjustment,
  show.axes.numbers = TRUE,
  show.grid.lines = !grepl("umap|tsne", tolower(reduction.use)),
  main = "make",
  sub = NULL,
  xlab = "make",
  ylab = "make",
  theme = theme_bw(),
  do.contour = FALSE,
  contour.color = "black",
  contour.linetype = 1,
  min.density = NA,
  max.density = NA,
  min.color = "#F0E442",
  max.color = "#0072B2",
```

```
  min.opacity = 0.2,
  max.opacity = 1,
  min = NA,
  max = NA,
  rename.color.groups = NULL,
  do.ellipse = FALSE,
  do.label = FALSE,
  labels.size = 5,
  labels.highlight = TRUE,
  labels.repel = TRUE,
  labels.split.by = split.by,
  labels.repel.adjust = list(),
  add.trajectory.lineages = NULL,
  add.trajectory.curves = NULL,
  trajectory.cluster.meta,
  trajectory.arrow.size = 0.15,
  data.out = FALSE,
  legend.show = TRUE,
  legend.color.title = "make",
  legend.color.breaks = waiver(),
  legend.color.breaks.labels = waiver(),
  legend.density.title = if (isBulk(object)) "Samples" else "Cells",
  legend.density.breaks = waiver(),
  legend.density.breaks.labels = waiver()
)

dittoScatterHex(
  object,
  x.var,
  y.var,
  color.var = NULL,
  bins = 30,
  color.method = NULL,
  split.by = NULL,
  extra.vars = NULL,
  cells.use = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  multivar.split.dir = c("col", "row"),
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  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,
    swap.rownames = NULL,
    min.density = NA,
    max.density = NA,
    min.color = "#F0E442",
    max.color = "#0072B2",
    min.opacity = 0.2,
    max.opacity = 1,
    min = NA,
    max = NA,
    rename.color.groups = NULL,
    xlab = x.var,
    ylab = y.var,
    main = "make",
    sub = NULL,
    theme = theme_bw(),
    do.contour = FALSE,
    contour.color = "black",
    contour.linetype = 1,
    do.ellipse = FALSE,
    do.label = FALSE,
    labels.size = 5,
    labels.highlight = TRUE,
    labels.repel = TRUE,
    labels.split.by = split.by,
    labels.repel.adjust = list(),
    add.trajectory.lineages = NULL,
    add.trajectory.curves = NULL,
    trajectory.cluster.meta,
    trajectory.arrow.size = 0.15,
    legend.show = TRUE,
    legend.color.title = "make",
    legend.color.breaks = waiver(),
    legend.color.breaks.labels = waiver(),
    legend.density.title = if (isBulk(object)) "Samples" else "Cells",
    legend.density.breaks = waiver(),
    legend.density.breaks.labels = waiver(),
    data.out = FALSE
  )
```

Arguments

`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment 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`.

`bins` Numeric or numeric vector giving the number of haxagonal bins in the x and y directions. Set to 30 by default.

`color.method` Works differently depending on whether the `color.var`-data is continuous versus discrete:

Continuous: String naming a function for how target data should be summarized for each bin. Can be any function that summarizes a numeric vector input with a single numeric output value. Default is `median`. Other useful options are `sum`, `mean`, `sd`, or `mad`. You can also use a previously assigned function; e.g. first run `pNonZero <- function(x) { sum(x!=0)/length(x)}`, then you give `color.method = "pNonZero"`

Discrete: A string signifying whether the color should (default) be simply based on the "max" grouping of the bin, based on "prop.<value>" the proportion of a specific value (e.g. "prop.A" or "prop.TRUE"), or based on the "max.prop"ortion of cells/samples belonging to any grouping.

`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 with "umap", "tsne", or "pca" within its name, (priority: UMAP > t-SNE > PCA) or the first dimensionality reduction slot if none of those exist.

Alternatively, a matrix (or data.frame) containing the dimensionality reduction embeddings themselves. The matrix should have as many rows as there are cells/samples in the `object`. Note that `dim.1` and `dim.2` will still be used to select which columns to pull from, and column names will serve as the default `xlab` & `ylab`.

`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 OR an integer vector specifying the indices of cells/samples 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.

`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.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.

`multivar.split.dir`
"row" or "col", sets the direction of faceting used for 'var' values when `var` is given multiple genes or metadata, and when `split.by` is used to provide additional data to facet by.

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

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. 'list(scales = "free")'.

For options, when giving 1 metadata to `split.by`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

`assay`, `slot` single strings or integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`adjustment` When plotting gene / feature expression, 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]

`swap.rownames` optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

`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 = FALSE for umap and tsne `reduction.use`, TRUE otherwise.

`main` String, sets the plot title. The default title is either "Density", `color.var`, or NULL, depending on the identity of `color.var`. To remove, set to `NULL`.

`sub` String, sets the plot subtitle.

`xlab`, `ylab` Strings which set the labels for the axes. 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.

do.contour
: Logical. Whether density-based contours should be displayed.

contour.color
: String that sets the color(s) of the do.contour contours.

contour.linetype
: String or numeric which sets the type of line used for do.contour contours. Defaults to "solid", but see linetype for other options.

min.density, max.density
: Number which sets the min/max values used for the density scale. Used no matter whether density is represented through opacity or color.

min.color, max.color
: color for the min/max values of the color scale.

min.opacity, max.opacity
: Scalar between [0,1] which sets the minimum or maximum opacity used for the density legend (when color is used for color.var data and density is shown via opacity).

min, max
: Number which sets the values associated with the minimum or maximum color for color.var data.

rename.color.groups
: String vector containing new names for the identities of discrete color groups.

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.

labels.split.by
: String of one or two metadata names which controls the facet-split calculations for label placements. Defaults to split.by, so generally there is no need to adjust this except when you are utilizing the extra.vars input to achieve manual faceting control.

labels.repel.adjust
: A named list which allows extra parameters to be pushed through to ggrepel function calls. List elements should be valid inputs to the geom_label_repel by default, or geom_text_repel when labels.highlight = FALSE.

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 provide add.trajectory.lineages = slingLineages('object').

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, (for dittoDimHex only, but not dittoScatterHex) 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 = slingCurves('object')`

`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 ("plot"), and data.frame of the underlying data for target cells ("data").

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

`legend.density.title`, `legend.color.title`
: Strings which set the title for the legends.

`legend.density.breaks`, `legend.color.breaks`
: Numeric vector which sets the discrete values to label in the density and color.var legends.

`legend.density.breaks.labels`, `legend.color.breaks.labels`
: String vector, with same length as `legend.*.breaks`, which sets the labels for the tick marks or hex icons of the associated legend.

`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`.

`assay.x`, `assay.y`, `assay.color`, `assay.extra`, `slot.x`, `slot.y`, `slot.color`, `slot.extra`
: single strings or integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use for each given data target. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`adjustment.x`, `adjustment.y`, `adjustment.color`, `adjustment.extra`
: For the given data target, when targeting gene / feature expression, 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]

Details

The functions create a dataframe with x and y coordinates for each cell/sample, determined by either `x.var` and `y.var` for `dittoScatterHex`, or `reduction.use`, `dim.1` (x), and `dim.2` (y) for `dittoDimHex`. Extra data requested by `color.var` for coloring, `split.by` for faceting, or

extra.var for manual external manipulations, are added to the dataframe as well. For expression/counts data, assay, slot, and adjustment inputs can be used to select which values to use, and if they should be adjusted in some way.

The dataframe is then subset to only target cells/samples based on the cells.use input.

Finally, a hex plot is created using this dataframe:

If color.var is not rovided, coloring is based on the density of cells/samples within each hex bin. When color.var is provided, density is represented through opacity while coloring is based on a summarization, chosen with the color.method input, of the target color.var data.

If split.by was used, the plot will be split into a matrix of panels based on the associated groupings.

Value

A ggplot object where colored hexagonal bins are used to summarize RNAseq data in a scatterplot or tSNE, PCA, UMAP.

Alternatively, if data.out=TRUE, a list containing two slots is output: the plot (named 'plot'), and a data.table containing the underlying data for target cells (named 'data').

Functions

- dittoDimHex(): Show RNAseq data overlayed on a tsne, pca, or similar, grouped into hexagonal bins
- dittoScatterHex(): Make a scatter plot of RNAseq data, grouped into hexagonal bins

Many characteristics of the plot can be adjusted using discrete inputs

- Colors: min.color and max.color adjust the colors for continuous data.
- For discrete color.var plotting with color.method = "max", colors are instead adjusted with color.panel and/or colors & the labels of the groupings can be changed using rename.color.groups.
- Titles and axes labels can be adjusted with main, sub, xlab, ylab, and legend.color.title and legend.density.title arguments.
- Legends can also be adjusted in other ways, using variables that all start with "legend." for easy tab completion lookup.

Additional Features

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.contour is provided, density gradiant contour lines will be overlaid with color and linetype adjustable via contour.color and contour.linetype.
- 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 with some code adapted from Giuseppe D'Agostino

See Also

`dittoDimPlot` and `dittoScatterPlot` for making very similar data representations, but where each cell is represented individually. It is often best to investigate your data with both the individual and hex-bin methods, then pick whichever is the best representation for your particular goal.

`getGenes` and `getMetas` to see what the `var`, `split.by`, etc. options are of an `object`.

`getReductions` to see what the `reduction.use` options are of an `object`.

Examples

```
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)))

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

# We don't have too many samples here, so let's increase the bin size.
dittoDimHex(myRNA, bins = 10)

# x and y bins can be set separately, useful for non-square plots
dittoDimHex(myRNA, bins = c(20, 10))

### Coloring
# Default coloring, as above, is by cell/sample density in the region, but
# 'color.var' can be used to color the data by another metric.
# Density with then be represented via bin opacity.
dittoDimHex(myRNA, color.var = "clustering", bins = 10)
dittoDimHex(myRNA, color.var = "gene1", bins = 10)

# 'color.method' is then used to adjust how the target data is summarized
dittoDimHex(myRNA, color.var = "groups", bins = 10,
    color.method = "max.prop")
dittoDimHex(myRNA, color.var = "gene1", bins = 10,
    color.method = "mean")
```

```
# One particularly useful 'color.method' for discrete 'color.var'-data is
#   to use 'prop.<value>' to color by the proportion of a particular value
#   within each bin:
dittoDimHex(myRNA, color.var = "groups", bins = 10,
    color.method = "prop.A")

### Additional Features:

# Faceting with 'split.by'
dittoDimHex(myRNA, bins = 10, split.by = "groups")
dittoDimHex(myRNA, bins = 10, split.by = c("groups", "clustering"))

# Faceting can also be used to show multiple continuous variables side-by-side
#   by giving a vector of continuous metadata or gene names to 'color.var'.
#   This can also be combined with 1 'split.by' variable, with direction then
#   controlled via 'multivar.split.dir':
dittoDimHex(myRNA, bins = 10,
    color.var = c("gene1", "gene2"))
dittoDimHex(myRNA, bins = 10,
    color.var = c("gene1", "gene2"),
    split.by = "groups")
dittoDimHex(myRNA, bins = 10,
    color.var = c("gene1", "gene2"),
    split.by = "groups",
    multivar.split.dir = "row")

# Underlying data output with 'data.out = TRUE'
dittoDimHex(myRNA, data.out = TRUE)

# Contour lines can be added with 'do.contours = TRUE'
dittoDimHex(myRNA, bins = 10,
    do.contour = TRUE,
    contour.color = "lightblue", # Optional, black by default
    contour.linetype = "dashed") # Optional, solid by default

# Trajectories can be added to dittoDimHex plots
dittoDimHex(myRNA, bins = 10,
    add.trajectory.lineages = list(c(1,2,4), c(1,4), c(1,3)),
    trajectory.cluster.meta = "clustering")
```

`dittoPlot` *Plots continuous data for customizeable cells'/samples' groupings on a y- (or x-) axis*

Description

Plots continuous data for customizeable cells'/samples' groupings on a y- (or x-) 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"),
  multivar.aes = c("split", "group", "color"),
  multivar.split.dir = c("col", "row"),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  swap.rownames = 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 = NA,
  max = NA,
  xlab = "make",
  x.labels = NULL,
  x.labels.rotate = NA,
  x.reorder = NULL,
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  do.raster = FALSE,
  raster.dpi = 300,
  jitter.size = 1,
  jitter.width = 0.2,
  jitter.color = "black",
  jitter.shape.legend.size = NA,
  jitter.shape.legend.show = TRUE,
  jitter.position.dodge = boxplot.position.dodge,
  boxplot.width = 0.2,
  boxplot.color = "black",
  boxplot.show.outliers = NA,
  boxplot.outlier.size = 1.5,
```

```
  boxplot.fill = TRUE,
  boxplot.position.dodge = vlnplot.width,
  boxplot.lineweight = 1,
  vlnplot.lineweight = 1,
  vlnplot.width = 1,
  vlnplot.scaling = "area",
  vlnplot.quantiles = NULL,
  ridgeplot.lineweight = 1,
  ridgeplot.scale = 1.25,
  ridgeplot.ymax.expansion = NA,
  ridgeplot.shape = c("smooth", "hist"),
  ridgeplot.bins = 30,
  ridgeplot.binwidth = NULL,
  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, SingleCellExperiment, or SummarizedExperiment object.
`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. Alternatively, a string vector naming multiple genes or metadata. This is the primary 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 supersets or subgroups when wanted, but it defaults to `group.by` so this input can be skipped otherwise.
`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' for bulk data) names OR an integer vector specifying the indices of cells/samples 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.

`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.

`multivar.aes` "split", "group", or "color", the plot feature to utilize for displaying 'var' value when `var` is given multiple genes or metadata. When set to "split", inputs `split.nrow`, `split.ncol`, and `split.adjust` can be used to

`multivar.split.dir`

"row" or "col", sets the direction of faceting used for 'var' values when `var` is given multiple genes or metadata, when `multivar.aes = "split"`, and when `split.by` is used to provide additional data to facet by.

`assay, slot` single strings or integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`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]

`swap.rownames` optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

`do.hover` Logical. Default = `FALSE`. If set to `TRUE`: 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 (assuming that there is a "jitter" in `plots`),

`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 display. Default = NA, which allows ggplot to set these limits based on the range of all data being shown.

`xlab` String which sets the grouping-axis label (=x-axis for box and violin plots, y-axis for ridgeplots). Set to `NULL` to remove.

`x.labels` String vector, c("label1","label2","label3",...) which overrides the names of groupings.

`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.

Recommendation for advanced users: If you find yourself coming back to this input too many times, an alternative solution that can be easier long-term is to make the target data into a factor, and to put its levels in the desired order: `factor(data, levels = c("level1", "level2", ...))`. `metaLevels` can be used to quickly get the identities that need to be part of this 'levels' input.

`split.nrow`, `split.ncol`
Integers which set the dimensions of faceting/splitting when a single metadata is given to `split.by`, or when multiple genes/metadata are given to `var` and `multivar.aes = "split"`.

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. 'list(scales = "free")'.

For options, when giving 1 metadata to `split.by` or if faceting is by a set of `vars`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

`do.raster` Logical. When set to `TRUE`, rasterizes the jitter plot layer, changing it from individually encoded points to a flattened set of pixels. This can be useful for editing in external programs (e.g. Illustrator) when there are many thousands of data points.

`raster.dpi` Number indicating dots/pixels per inch (dpi) to use for rasterization. Default = 300.

`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.

Note for when `color.by` is used to split x-axis groupings into additional bins: ggplot does not shrink jitter widths accordingly, so be sure to do so yourself! Ideally, needs to be 0.5/num_subgroups.

`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`.

`jitter.position.dodge`
Scalar which adjusts the relative distance between jitter widths when multiple subgroups exist per `group.by` grouping (a.k.a. when `group.by` and `color.by` are not equal). Similar to `boxplot.position.dodge` input & defaults to the value of that input so that BOTH will actually be adjusted when only, say, `boxplot.position.dodge = 0.3` is given.

`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.outlier.size`
Scalar which adjusts the size of points used to mark outliers

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

`boxplot.position.dodge`
Scalar which adjusts the relative distance between boxplots when multiple are drawn per grouping (a.k.a. when `group.by` and `color.by` are not equal). By default, this input actually controls the value of `jitter.position.dodge` unless the `jitter` version is provided separately.

`boxplot.lineweight`
Scalar which adjusts the thickness of boxplot lines.

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

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

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

`vlnplot.quantiles`
: Single number or numeric vector of values in [0,1] naming quantiles at which to draw a horizontal line within each violin plot. Example: `c(0.1, 0.5, 0.9)`

`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

`ridgeplot.ymax.expansion`
: Scalar which adjusts the minimal space between the top-most grouping and the top of the plot in order to ensure that the curve is not cut off by the plotting grid. The larger the value, the greater the space requested. When left as NA, dittoSeq will attempt to determine an ideal value itself based on the number of groups & linear interpolation between these goal posts: 0.6 when g<=3, 0.1 when g==12, and 0.05 when g>=34, where g is the number of groups.

`ridgeplot.shape`
: Either "smooth" or "hist", sets whether ridges will be smoothed (the typical, and default) versus rectangular like a histogram. (Note: as of the time shape "hist" was added, combination of jittered points is not supported by the `stat_binline` that dittoSeq relies on.)

`ridgeplot.bins`
: Integer which sets how many chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Overridden by `ridgeplot.binwidth` when that input is provided.

`ridgeplot.binwidth`
: Integer which sets the width of chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Takes precedence over `ridgeplot.bins` when provided.

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

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

`...`
: 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 `"vlnplot"` presence 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 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 ggplot where additional data will be displayed when the cursor is hovered over jitter points.

Functions

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

Many characteristics of the plot can be adjusted using discrete inputs

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. For example, all jitter adjustments start with `"jitter."`, such as `jitter.size` and `jitter.width`.

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

Additionally:

- **Colors can be adjusted** with `color.panel`.
- **Subgroupings:** `color.by` can be utilized to split major `group.by` groupings into subgroups. When this is done in y-axis plotting, dittoSeq automatically ensures the centers of all geoms will align, but users will need to manually adjust `jitter.width` to less than 0.5/num_subgroups to avoid overlaps. There are also three inputs through which one can use to control geom-center placement, but the easiest way to do all at once so is to just adjust `vlnplot.width`! The other two: `boxplot.position.dodge`, and `jitter.position.dodge`.
- **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.
- **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 = FALSE`.
- **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 on/off with `x.labels.rotate = TRUE/FALSE`.
- **Shapes used** in conjunction with `shape.by` can be adjusted with `shape.panel`.
- 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

```
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")
dittoPlot(object = myRNA, var = "gene1", group.by = "conditions",
    color.by = "timepoint")
```

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

### Provided wrappers enable certain easy adjustments of the 'plots' parameter.
# Quickly make a Boxplot
dittoBoxPlot(myRNA, "gene1", group.by = "timepoint")
# Quickly make a Ridgeplot, with or without jitter
dittoRidgePlot(myRNA, "gene1", group.by = "timepoint")
dittoRidgeJitter(myRNA, "gene1", group.by = "timepoint")

### Additional Functionality
# 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

# Multiple genes or continuous metadata can also be plotted by giving them as
#   a vector to 'var'. One aesthetic of the plot will then be used to display
#   'var'-info, and you can control which (faceting / "split", x-axis grouping
#   / "group", or color / "color") with 'multivar.aes':
dittoPlot(object = myRNA, group.by = "timepoint",
    var = c("gene1", "gene2"))
dittoPlot(object = myRNA, group.by = "timepoint",
    var = c("gene1", "gene2"),
    multivar.aes = "group")
dittoPlot(object = myRNA, group.by = "timepoint",
    var = c("gene1", "gene2"),
    multivar.aes = "color")

# 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")
```

dittoPlotVarsAcrossGroups

Generates a dittoPlot where data points are genes/metadata summaries, per groups, instead of individual values per cells/samples.

Description

Generates a dittoPlot where data points are genes/metadata summaries, per groups, instead of individual values per cells/samples.

Usage

```
dittoPlotVarsAcrossGroups(
  object,
  vars,
  group.by,
  color.by = group.by,
  split.by = NULL,
  summary.fxn = mean,
  cells.use = NULL,
  plots = c("vlnplot", "jitter"),
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = "z-score",
  swap.rownames = NULL,
  do.hover = FALSE,
  main = NULL,
  sub = NULL,
  ylab = "make",
  y.breaks = NULL,
  min = NA,
  max = NA,
  xlab = group.by,
  x.labels = NULL,
  x.labels.rotate = NA,
  x.reorder = NULL,
  groupings.drop.unused = TRUE,
  color.panel = dittoColors(),
  colors = c(seq_along(color.panel)),
  theme = theme_classic(),
  jitter.size = 1,
  jitter.width = 0.2,
  jitter.color = "black",
  jitter.position.dodge = boxplot.position.dodge,
  do.raster = FALSE,
  raster.dpi = 300,
  boxplot.width = 0.2,
```

```
  boxplot.color = "black",
  boxplot.show.outliers = NA,
  boxplot.outlier.size = 1.5,
  boxplot.fill = TRUE,
  boxplot.position.dodge = vlnplot.width,
  boxplot.lineweight = 1,
  vlnplot.lineweight = 1,
  vlnplot.width = 1,
  vlnplot.scaling = "area",
  vlnplot.quantiles = NULL,
  ridgeplot.lineweight = 1,
  ridgeplot.scale = 1.25,
  ridgeplot.ymax.expansion = NA,
  ridgeplot.shape = c("smooth", "hist"),
  ridgeplot.bins = 30,
  ridgeplot.binwidth = NULL,
  add.line = NULL,
  line.linetype = "dashed",
  line.color = "black",
  split.nrow = NULL,
  split.ncol = NULL,
  split.adjust = list(),
  legend.show = TRUE,
  legend.title = NULL,
  data.out = FALSE
)
```

Arguments

`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment 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.
`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`
`summary.fxn`	A function which sets how variables' data will be summarized across 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 examples: `median`, `max`, or `function(x) mean(x!=0)`.

`cells.use` String vector of cells'/samples' names OR an integer vector specifying the indices of cells/samples 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.

`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 integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

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

`swap.rownames` optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

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

`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 the 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 grid lines. 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 display. Default = NA, which allows ggplot to set these limits based on the range of all data being shown.

`xlab` String which sets the grouping-axis label (=x-axis for box and violin plots, y-axis for ridgeplots). Set to `NULL` to remove.

`x.labels` String vector, c("label1","label2","label3",...) which overrides the names of groupings.

`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.

Recommendation for advanced users: If you find yourself coming back to this input too many times, an alternative solution that can be easier long-term is to make the target data into a factor, and to put its levels in the desired order: `factor(data, levels = c("level1", "level2", ...))`. `metaLevels` can be used to quickly get the identities that need to be part of this 'levels' input.

`groupings.drop.unused`
Logical. `TRUE` by default. If `group.by`-data is a factor, factor levels are retained for ordering purposes, but some level(s) can end up with zero cells left after `cells.use` subsetting. By default, we remove them, but you can set this input to `FALSE` to keep them.

`color.panel` String vector which sets the colors to draw from for plot fills.

`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.

Note for when `color.by` is used to split x-axis groupings into additional bins: ggplot does not shrink jitter widths accordingly, so be sure to do so yourself! Ideally, needs to be 0.5/num_subgroups.

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

`jitter.position.dodge`
Scalar which adjusts the relative distance between jitter widths when multiple subgroups exist per `group.by` grouping (a.k.a. when `group.by` and `color.by` are not equal). Similar to `boxplot.position.dodge` input & defaults to the value of that input so that BOTH will actually be adjusted when only, say, `boxplot.position.dodge = 0.3` is given.

`do.raster` Logical. When set to `TRUE`, rasterizes the jitter plot layer, changing it from individually encoded points to a flattened set of pixels. This can be useful for editing in external programs (e.g. Illustrator) when there are many thousands of data points.

`raster.dpi` Number indicating dots/pixels per inch (dpi) to use for rasterization. Default = 300.

`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.outlier.size`
: Scalar which adjusts the size of points used to mark outliers

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

`boxplot.position.dodge`
: Scalar which adjusts the relative distance between boxplots when multiple are drawn per grouping (a.k.a. when `group.by` and `color.by` are not equal). By default, this input actually controls the value of `jitter.position.dodge` unless the `jitter` version is provided separately.

`boxplot.lineweight`
: Scalar which adjusts the thickness of boxplot lines.

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

`vlnplot.width`
: Scalar which sets the width/spread of violin plots in the x direction

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

`vlnplot.quantiles`
: Single number or numeric vector of values in [0,1] naming quantiles at which to draw a horizontal line within each violin plot. Example: `c(0.1, 0.5, 0.9)`

`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

`ridgeplot.ymax.expansion`
: Scalar which adjusts the minimal space between the top-most grouping and the top of the plot in order to ensure that the curve is not cut off by the plotting grid. The larger the value, the greater the space requested. When left as NA, dittoSeq will attempt to determine an ideal value itself based on the number of groups & linear interpolation between these goal posts: 0.6 when g<=3, 0.1 when g==12, and 0.05 when g>=34, where g is the number of groups.

`ridgeplot.shape`
: Either "smooth" or "hist", sets whether ridges will be smoothed (the typical, and default) versus rectangular like a histogram. (Note: as of the time shape "hist" was added, combination of jittered points is not supported by the `stat_binline` that dittoSeq relies on.)

`ridgeplot.bins`
: Integer which sets how many chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Overridden by `ridgeplot.binwidth` when that input is provided.

`ridgeplot.binwidth`
: Integer which sets the width of chunks to break the x-axis into when `ridgeplot.shape = "hist"`. Takes precedence over `ridgeplot.bins` when provided.

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

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

`split.adjust` A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. ‘list(scales = "free")‘.

For options, when giving 1 metadata to `split.by`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

`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.

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

Details

Generally, this function will output a dittoPlot where each data point represents a gene (or metadata) rather than a cell/sample. Values are the summary (`mean` by default) of the values for each gene or metadata requested with `vars`, within each group set by `group.by`.

To start with, 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 object

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. For example, all jitter adjustments start with `"jitter."`, such as `jitter.size` and `jitter.width`.

Inclusion of "ridgeplot" overrides "boxplot" and "vlnplot" presence and changes the plot to be horizontal.

Additionally:

- **Colors can be adjusted** with color.panel.
- **Subgroupings:** color.by can be utilized to split major group.by groupings into subgroups. When this is done in y-axis plotting, dittoSeq automatically ensures the centers of all geoms will align, but users will need to manually adjust jitter.width to less than 0.5/num_subgroups to avoid overlaps. There are also three inputs through which one can use to control geom-center placement, but the easiest way to do all at once so is to just adjust vlnplot.width! The other two: boxplot.position.dodge, and jitter.position.dodge.
- **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.
- **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 = FALSE.
- **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 on/off with x.labels.rotate = TRUE/FALSE.
- **Shapes used** in conjunction with shape.by can be adjusted with shape.panel.

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.

dittoDotPlot for an alternative representation of per-group summaries of multiple vars where all vars are displayed separately, but still in a single plot.

Examples

```
example(importDittoBulk, echo = FALSE)

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

dittoPlotVarsAcrossGroups(
    myRNA, 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(myRNA, 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(myRNA, genes, "timepoint",
    plots = c("vlnplot","jitter","boxplot"),
    boxplot.color = "white",
    boxplot.fill = FALSE)

## Data can be summarized in other ways by changing the summary.fxn input.
#  median
dittoPlotVarsAcrossGroups(myRNA, genes, "timepoint",
    summary.fxn = median,
    adjustment = NULL)
#  Percent non-zero expression ( = boring for this fake data)
percent <- function(x) {sum(x!=0)/length(x)}
dittoPlotVarsAcrossGroups(myRNA, genes, "timepoint",
    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(
        myRNA, genes, "timepoint",
        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,
  multivar.split.dir = c("col", "row"),
  show.others = FALSE,
  split.show.all.others = TRUE,
  size = 1,
```

```
opacity = 1,
color.panel = dittoColors(),
colors = seq_along(color.panel),
split.nrow = NULL,
split.ncol = NULL,
split.adjust = list(),
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,
swap.rownames = 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 = NA,
max = NA,
order = c("unordered", "increasing", "decreasing", "randomize"),
xlab = x.var,
ylab = y.var,
main = "make",
sub = NULL,
theme = theme_bw(),
do.hover = FALSE,
hover.data = NULL,
hover.assay = .default_assay(object),
hover.slot = .default_slot(object),
hover.adjustment = NULL,
do.contour = FALSE,
contour.color = "black",
contour.linetype = 1,
add.trajectory.lineages = NULL,
add.trajectory.curves = NULL,
trajectory.cluster.meta,
trajectory.arrow.size = 0.15,
do.letter = FALSE,
do.ellipse = FALSE,
do.label = FALSE,
labels.size = 5,
```

```
  labels.highlight = TRUE,
  labels.repel = TRUE,
  labels.split.by = split.by,
  labels.repel.adjust = list(),
  legend.show = TRUE,
  legend.color.title = "make",
  legend.color.size = 5,
  legend.color.breaks = waiver(),
  legend.color.breaks.labels = waiver(),
  legend.shape.title = shape.by,
  legend.shape.size = 5,
  do.raster = FALSE,
  raster.dpi = 300,
  data.out = FALSE
)
```

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment 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 OR an integer vector specifying the indices of cells/samples 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.

`multivar.split.dir`
: "row" or "col", sets the direction of faceting used for 'var' values when `var` is given multiple genes or metadata, and when `split.by` is used to provide additional data to facet by.

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

`split.show.all.others`
: Logical which sets whether gray "others" cells of facets should include all cells of other facets (`TRUE`) versus just cells left out by `cell.use` (`FALSE`).

`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`.

`split.adjust`
: A named list which allows extra parameters to be pushed through to the faceting function call. List elements should be valid inputs to the faceting functions, e.g. 'list(scales = "free")'.

  For options, when giving 1 metadata to `split.by`, see `facet_wrap`, OR when giving 2 metadatas to `split.by`, see `facet_grid`.

`assay.x`, `assay.y`, `assay.color`, `assay.extra`, `slot.x`, `slot.y`, `slot.color`, `slot.extra`
: single strings or integers (SCEs and SEs) or an optionally named vector of such values that set which expression data to use for each given data target. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.

`adjustment.x`, `adjustment.y`, `adjustment.color`, `adjustment.extra`
: For the given data target, when targeting gene / feature expression, 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]

`swap.rownames`
: optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

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 min value of color.var data. Default = yellow

max.color
: color for max value of color.var data. Default = blue

min, max
: Number which sets the values associated with the minimum or maximum colors.

order
: String. If the data should be plotted based on the order of the color data, sets whether to plot (from back to front) in "increasing", "decreasing", "randomize" order. If left as "unordered", plot order is simply based on the order of cells within the object.

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.

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.

do.contour
: Logical. Whether density-based contours should be displayed.

contour.color
: String that sets the color(s) of the do.contour contours.

contour.linetype
: String or numeric which sets the type of line used for do.contour contours. Defaults to "solid", but see linetype for other options.

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 provide add.trajectory.lineages = slingLineages('object').

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

`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.

`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.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.

`labels.split.by`
: String of one or two metadata names which controls the facet-split calculations for label placements. Defaults to `split.by`, so generally there is no need to adjust this except when you are utilizing the `extra.vars` input to achieve manual faceting control.

`labels.repel.adjust`
: A named list which allows extra parameters to be pushed through to ggrepel function calls. List elements should be valid inputs to the `geom_label_repel` by default, or `geom_text_repel` when `labels.highlight = FALSE`.

`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 label in the color-scale legend for continuous data.

`legend.color.breaks.labels`
: String vector, with same length as `legend.breaks`, which sets the labels for the tick marks of the color-scale.

do.raster	Logical. When set to TRUE, rasterizes the internal plot layer, changing it from individually encoded points to a flattened set of pixels. This can be useful for editing in external programs (e.g. Illustrator) when there are many thousands of data points.
raster.dpi	Number indicating dots/pixels per inch (dpi) to use for rasterization. Default = 300.
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").

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 and Jared Andrews

See Also

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

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

dittoDimHex and dittoScatterHex for showing very similar data representations, but where nearby cells are summarized together in hexagonal bins.

Examples

```
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 = "gene1", y.var = "gene2",
    color.var = "groups",
    shape.by = "SNP",
    size = 3)
dittoScatterPlot(
    myRNA, x.var = "gene1", y.var = "gene2",
    color.var = "gene3")

# Note: scatterplots like this can be very useful for dataset QC, especially
#   with percentage of mitochondrial reads as the color overlay.
dittoScatterPlot(myRNA,
    x.var = "nCount_RNA", y.var = "nFeature_RNA",
    color.var = "percent.mito")

# 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
```

```
dittoScatterPlot(myRNA, x.var = "gene1", y.var = "gene2",
    extra.vars = c("SNP")) +
    facet_wrap("SNP", ncol = 1, strip.position = "left")

# Countours can also be added to help illuminate overlapping samples
dittoScatterPlot(myRNA, x.var = "gene1", y.var = "gene2",
    do.contour = TRUE)

# Multiple continuous metadata or genes can also be plotted together by
#   giving that vector to 'color.var':
dittoScatterPlot(myRNA, x.var = "gene1", y.var = "gene2",
    color.var = c("gene3", "gene4"))
# This functionality can be combined with 1 additional 'split.by' variable,
#   with the directionality then controlled via 'multivar.split.dir':
dittoScatterPlot(myRNA, x.var = "gene1", y.var = "gene2",
    color.var = c("gene3", "gene4"),
    split.by = "timepoint",
    multivar.split.dir = "col")
dittoScatterPlot(myRNA, x.var = "gene1", y.var = "gene2",
    color.var = c("gene3", "gene4"),
    split.by = "timepoint",
    multivar.split.dir = "row")
```

`dittoSeq` *dittoSeq*

Description

This package was built to make the visualization of single-cell and bulk RNA-sequencing data pipeline-agnostic and accessible for both experienced and novice coders, and for color vision impaired individuals.

Details

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

Default `dittoColors` are color blindness friendly and adapted from Wong B, "Points of view: Color blindness." Nature Methods, 2011.

To report bugs, suggest new features, or ask for help, the best method is to create an issue on the github, here, or the bioconductor support site (be sure to tag 'dittoSeq' so that I get a notification!), here

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,
  adj.fxn = NULL,
  swap.rownames = NULL
)
```

Arguments

`gene`	quoted "gene" name = REQUIRED. the gene whose expression data should be retrieved.
`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment object.
`assay, slot`	single strings or integers (SCEs and SEs) or a vector of such values that set which expression data to use. See `GeneTargeting` for specifics and examples – Seurat and SingleCellExperiment objects deal with these differently, and functionality additions in dittoSeq have led to some minimal divergence from the native methodologies.
`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]
`adj.fxn`	A function which takes a vector (of metadata values) and returns a vector of the same length. For example, `function(x) {log2(x)}` or `as.factor`
`swap.rownames`	optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

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")

# Log2'd
gene("gene1", object = myRNA, assay = "counts",
    adj.fxn = function(x) {log2(x)})

# 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")
}
```

GeneTargeting *Control of Gene/Feature targeting*

Description

Control of Gene/Feature targeting

Overview

As of dittoSeq version 1.15.2, we made it possible to target genes / features from across multiple modalities. Here, we describe intricacies of how 'assay', 'slot', and 'swap.rownames' inputs now work to allow for this purpose.

Control of gene/feature targeting in dittoSeq functions aims to blend seamlessly with how similar control works in Seurat, SingleCellExperiment (SCE), and other packages that deal with these data structures. However, as we've built in new features into dittoSeq, and the Seurat and SCE-package maintainers extend their tools as well, some divergence was to be expected.

The way Seurat and SingleCellExperiment objects hold data from multiple modalities is quite distinct, thus it is worth describing each distinctly.

It's also important to note, that *both structures utilize the term 'assay', but they utilize it for distinct meanings.* Keep that in mind because we chose to stick with the native terminologies within dittoSeq in order maintain intuitiveness with other Seurat or SCE data accession methods. In other words, rather than enforcing a new consistent paradigm, the native Seurat 'assay' meaning is respected for Seurat objects, and the native SCE 'assay' meaning is respected for SCE objects.

Defaults

When not provided by the user, the defaults for `assay` and `slot` inputs are:

- Seurat-v3+: `assay` = `DefaultAssay(object)`, `slot` = "data"
- Seurat-v2 (v2 pre-dates Seurat's own multi-modal capabilities): `assay` is not used, `slot` = "data"
- SingleCellExperiment or SummarizedExperiment: `assay` = whichever of "logcounts", "normcounts", or "counts" are found to exist first, prioritized in that order, otherwise the first assay of object's top-level / primary modality; `slot` is not used.

The default for `swap.rownames` is `NULL`, a.k.a. not used.

Control of Gene/Feature targeting in Seurat objects

For Seurat objects, dittoSeq uses of its `assay` and `slot` inputs for gene/feature retrieval control, and ultimately makes use of Seurat's GetAssayData function for extracting data. (See: '?SeuratObject::GetAssayData')

To allow targeting of features across multiple modalities, we allow provision of multiple assay names to dittoSeq's version of the 'assay' input. Internally, dittoSeq will then loop through all values of 'assay', making a separate calls to GetAssayData for each assay.

Otherwise, dittoSeq's `assay` and `slot` inputs work exactly the same as described in Seurat's documentation.

Phrased another way, it works via inputs:

- `assay` - takes the name(s) of Seurat Assays to target. Examples: `"RNA"` or `c("RNA", "ADT")`
- `slot` - "counts", "data", or "scale.data". Directs which 'slot' of data from the targeted assays to extract from. Example: `"data"`

As an example, if you wanted to plot raw counts data from 1) the CD4 gene of the RNA assay and 2) the CD4.1 marker of an ADT assay, you would:

- 1. point the `var` or `vars` input of the plotter to `c("CD4", "CD4.1")`
- 2. target both modalities via `assay = c("RNA", "ADT")` (Note that "RNA" and "ADT" are the default assay names typically used, but you do need to match with what is in your own Seurat object if your assays are named differently.)
- 3. target the raw counts data via `slot = "counts"`

Control of Gene/Feature targeting in SingleCellExperiment objects

For SCE objects, dittoSeq makes use of its `assay` input for both modality and data form (the meaning of 'assay' for SCEs) control, and ultimately makes use of the `assay` and `altExp` functions for extracting data.

Additionally, we allow use of the `swap.rownames` input to allow targeting & display of alternative gene/feature names. The implementation here is that rownames of the extracted assay data are swapped out for the given `rowData` column of the object (or altExp). When used, note that you will need to use these swapped names for targeting genes / features with `gene`, `var`, or `vars` inputs.

In SCE objects themselves, the primary modality's expression data are stored in 'assay's of the SCE object. You might have one assay containing raw data, and another containing log-normalized data. Additional details of genes/features of this modality, possibly including alternative gene names, can be stored in the object's 'rowData' slot. When additional modalities are collected, the way to store them is via a nested SCE object called an "alternative experiment". Any number of these can be stored in the 'altExps' slot of the SCE object. Each alternative experiment can contain any number of assays. Again each will often have one representing raw data and another representing a normalized form of that data. And, these alternative experiments might also make use of their rowData to store additional characteristics or names of each gene/feature.

The system feels a bit more complicated here, because the SCE system is itself a bit more complicated. But the hope is that this system becomes simple to work with once learned!

To allow targeting of features across multiple modalities, dittoSeq's `assay` input can be given:

- Simplest form: a single string or string vector where values are either the names of an assay of the primary modality OR the name of an alternative experiment to target, with 'main' as an indicator for the primary modality and 'altexp' as a shortcut for indicating "the first altExp". In this form, when 'main', 'altexp', or the actual name of an alternative experiment are used, the first assay of that targeted modality will be used.
- Explicit form: a named string or named vector of string values where names indicate the modality/experiment to target and values indicate what assays of those experiments to target. Here again, you can use 'main' or 'altexp' as names to mean the primary modality and "the first altExp", respectively.
- These methods can also be combined. A few examples:
  - Using the simplified method only: `assay = c('main', 'altexp', 'hto')` will target the first assays each of the main object, of the first alternative experiment of the object, and also of an alternative experiment named 'hto'.
  - Using the explicit form only: `assay = c('main'='logexp', 'adt'='clr', 'altexp'='raw')` will target 1) the logexp-named assay of the main object, 2) the clr-named assay of an alternative experiment named 'adt', and 3) the raw-named assay of the first alternative experiment of the object.
  - Using a combination of the two: `assay = c('logexp', 'adt'='clr')` will target 1) the logexp-named assay of the primary modilty, unless there is an alternative experiment named 'logexp' which will lead to grabbing the first assay of that modality, and 2) the clr-named assay of an alternative experiment named 'adt'.

The `swap.rownames` input allows swapping to alternative names for genes/features via provision of a column name of rowData(object). The values of that rowData column are then used to identify and label features of the moadilty's assays instead of the original rownames of the assays. To allow

swap.rownames to also work with the multi-modality access system in the most simplified way, the swap.rownames input also has both a simple and an explicit provision system:

- Simple form: a single string or string vector where all modalities will be checked for the presence of a these values in colnames of their rowData. If multiple matches are found, priority goes to the earlier value. Values of matched rowData columns are then set (internally to dittoSeq only) as the rownames of the modality.
- Explicit form: similar to the explicit assay use, a named string or named vector of string values where names indicate the modality/experiment to target and values indicate columns to look for among the given modality's rowData. 'main' should be used as the name / indicator for the primary modality, and 'altexp' can be used as a shortcut for indicating "the first altExp".
- Examples:
  - Simplified1: Using `assay = c('main', 'altexp')`, `swap.rownames = "SYMBOL"` with an object where the primary modality rowData has a SYMBOL column and the first alternative experiment's rowData is empty, will lead to swapping to the SYMBOL values for main modality features and use of original rownames for the alternative experiment's features. (You will also see a warning indicating that the rownames were not swapped for the alternative experiment.)
  - Simplified2: Using `assay = c('main', 'altexp')`, `swap.rownames = "SYMBOL"` with an object where both modalities' rowData have a SYMBOL column, will lead to swapping to the SYMBOL values both modalities (and no warning).
  - Explicit: Using `assay = c('main', 'altexp')`, `swap.rownames = c(main="SYMBOL")` with an object where both modalities' rowData have a SYMBOL column, will lead to swapping to the SYMBOL values for main modality only.

As a full example, if you wanted to plot from 1) the raw 'counts' assay for a CD4 gene of the primary modality and 2) the normalized 'logexp' assay for a CD4.1 marker of an alternative experiment assay named 'ADT', but where 3) the rownames of these modalities are Ensembl ids while gene symbol names are held in a rowData column of both modalities that is named "symbols", the simplest provision method is:

- 1. point the `var` or `vars` input of the plotter to `c("CD4", "CD4.1")`
- 2. target the counts assay of the primary modality and logexp assay of the ADT alternative experiment via `assay = c('counts', ADT = 'logexp')`
- 3. swap to the symbol names of features from both modlities by also giving `swap.rownames = "symbols"`

Some edge-cases for SingleCellExperiment objects

Some choices within dittoSeq's multi-modality implementation for SCEs were made with a prioritization of ease over creation of edge-cases. Thus, a few known edge-cases exist:

- Avoid naming alternative experiments as 'main' or 'altexp'. Because these tokens have been chosen as indicators of "top-level data", and "the first alternative experiment", respectively, any alternative experiment given one of these names will not be able to be reliably accessed via dittoSeq's system.

- Explicit-path is required for top-level assays named 'altexp' Use `assay = c(main='altexp')` for a top-level assay named 'altexp'. Because we think the "simple path" is usefully simpler for cases where it works, `assay = 'altexp'` and `assay = c('main'='altexp')` are not equivalent. The explicit method MUST be used to extract from an assay named 'altexp' because `assay = 'altexp'` will instead target the first assay of the first altExp of the SCE.

Author(s)

Dan Bunis

`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), swap.rownames = NULL)
```

Arguments

`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment object.
`assay`	Single string or integer that sets which set of seq data inside the object to check.
`swap.rownames`	optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See `GeneTargeting` for more specifics and examples.

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 Seurat, SingleCellExperiment, or SummarizedExperiment 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 Seurat, SingleCellExperiment, or SummarizedExperiment 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 (SCE) 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, (3) a user-constructed data.frame created by reading in a demuxlet.best file.

Then it matches barcodes and adds demuxlet-information to the Seurat or SCE as metadata.

For a note on how best to utilize this function with multi-lane droplet-based data, see the devoted section below.

Specifically:

1. 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 "-#" at the end of cell names, as is the typical output of the 10X cellranger count & aggr pipeline.

(1a. If `demuxlet.best` was provided as a set of separate file locations (recommended usage in conjunction with 'cellranger aggr'), the "-#" at the ends of BARCODEs columns from these files are incremented on read-in so that they can match the incrementation applied by cellranger aggr. See the section on multi-lane scRNAseq for more.)

2. Barcodes in the demuxlet .best data are then matched to barcodes in the `object`. The cell names, `colnames(object)`, are used by default for this matching, but if these have been modified from what would have been given to demuxlet – outside of `-#` at the end or `***_`'s at the beginning, as can be added in common merge functions – `raw.cell.names` can be provided and these cell names used instead.

3. Singlet/doublet/ambiguous calls and sample identities (1st only for doublets) are parsed and carried into metadata.

4. 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`.

Value

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

Metadata Added

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.)

For data from multi-(droplet-gen-)lane scRNAseq

There are many different ways such data might initially be processed which will affect its accessibility to `importDemux()`.

Initial Processing: 10X recommends running cellranger counts individually for each well/lane. Non-10X droplet-based data from separate lanes should also be processed separately, at least for the steps of collecting reads for individual cells. NOT processing such droplet lanes separately will create artificial doublets from cells that ended up with similar barcodes, but in separate droplet-gen lanes. Thus, proper processing initially leads to creation of separate counts matrices for each droplet-generation lane.

Combining data from each lane: These per-lane counts matrices can be combined in various ways. All options will alter the cell barcode names in a way that makes them unique across lanes, but this uniquification is achieved varies.

Counts table combination methods generally do not adjust adjust BAM files – specifically the cell names embedded within the BAM files which is demuxlet uses for its BARCODEs column. Thus cell names data may needs to be modified in a proper way in order to make the `object`*'s cell names and* `demuxlet.best`*'s BARCODEs match.*

Running Demuxlet: Demuxlet should also be run, separately, on the BAM files of each individual lane. Imporperly running demuxlet on a combined BAM file can lead to loss of lane information and then to generation of artificial doublet calls for cells of distinct wells that received simiar barcodes. The BAM file associated with each demuxlet run is what is used for generating the BARCODE column of the demuxlet output.

How importDemux() handles barcode matching: `importDemux` is built to work with the 'cellranger aggr' pipeline by default, but can be used for demuxlet datasets processed differently as well (Option 2).

- Option 1: When you merge matrices of all lanes with **cellranger aggr before R import**, aggr's barcode uniquification method is to increment a "-1", "-2", "-3", ... "-#" that is appended to the end of all barcode names. The number is incremented for each succesive lane. Note that lane-numbers depend on the order in which they were supplied to cellranger aggr.
  - **to use:** Simply supply a `demuxlet.best` a vector containing the locations of the sepearate '.best' outputs for each lane, *in the same order that lanes were provided to aggr*. `importDemux` will adjust the "-#" in the `demuxlet.best` BARCODEs automatically before performing the matching step.
- Option 2: When you instead **import** your counts data into a Seurat or SingleCellExperiment, and **then merge** the separate objects into one, the uniquifiction method is dependent on your particular method.
  - **to use:** For these methods, it is easiest to 1) *import* your counts data, 2) transfer in your demuxlet info with *importDemux()* to each lane's object idividually (You can supply unique lane identifiers to the `lane.names` input.), and then 3) *merge* the separate objects.

- Extra notes for any alternative cases:
  - For Seurat's `merge()`, user-defined strings can be appended to the start of the barcodes, followed by an "_". By default, `importDemux()` will ignore these, but such ignorance can be controlled with the `trim.before_` input.
  - Alternatively, cell names that are consistent with the `demuxlet.best` BARCODEs can be supplied to the `raw.cell.names` input.

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 SingleCellExperiment format that will work with other dittoSeq functions.*

Description

import bulk sequencing data into a SingleCellExperiment format that will work with other dittoSeq functions.

Usage

```
importDittoBulk(x, reductions = NULL, metadata = NULL, combine_metadata = TRUE)
```

Arguments

`x`	A `DGEList`, or `SummarizedExperiment` (includes `DESeqDataSet`) class object containing the sequencing data to be imported. Alternatively, for import from a raw matrix format, a named list of matrices (or matrix-like objects) where names will become the assay names of the eventual SCE. NOTE: As of dittoSeq version 1.1.11, all dittoSeq functions can work directly with SummarizedExperiment objects, so this import function is nolonger required for such data.
`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` and `dittoDimHex`. For each matrix, rows of the matrices should represent the different samples of the dataset, and columns the different dimensions.

`metadata` A data.frame (or data.frame-like object) where rows represent samples and named columns represent the extra information about such samples that should be accessible to visualizations. The names of these columns can then be used to retrieve and plot such data in any dittoSeq visualization.

`combine_metadata`
Logical which sets whether original `colData` (DESeqDataSet/SummarizedExperiment) or `$samples` (DGEList) from `x` should be retained.

When `x` is a SummarizedExperiment or DGEList:

- When `FALSE`, sample metadata inside `x` (colData or $samples) is ignored entirely.
- When `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 take priority.**

Value

A `SingleCellExperiment` object...

that contains all assays (SummarizedExperiment; includes DESeqDataSets), all standard slots (DGEList; see below for specifics), or expression matrices of the input `x`, as well as any dimensionality reductions provided to `reductions`, and any provided `metadata` stored in colData.

Note about SummarizedExperiments

As of dittoSeq version 1.1.11, all dittoSeq functions can work directly with SummarizedExperiment objects, so this import function is nolonger required for such data.

Note on assay names

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 visualizations default to grabbing expression data from an assay named logcounts > normcounts > counts

See Also

`SingleCellExperiment` for more information about this storage structure.

Examples

```
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))
score1 <- seq_len(nsamples)/2
score2 <- rnorm(nsamples)

### We can import the counts directly
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 output 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,
        score1 = score1,
        score2 = score2),
    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 given object would be treated as bulk versus single-cell by dittoSeq*

Description

Retrieve whether a given object would be treated as bulk versus single-cell by dittoSeq

Usage

```
isBulk(object)
```

Arguments

`object`	A target Seurat, SingleCellExperiment, or SummarizedExperiment object

Value

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

- TRUE for SummarizedExperiments that are not SCEs, and for SCEs with `$bulk = TRUE` in their internal metadata.
- FALSE for any other object type and for SCEs without such internal metadata

See Also

setBulk to (add to and) set the internal metadata of an SCE to say whether the object repressents bulk data.

Examples

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

isBulk(myRNA)

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

---

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,
  swap.rownames = NULL
)
```

Arguments

test	String or vector of strings, the "potential.gene.name"(s) to check for.
object	A Seurat, SingleCellExperiment, or SummarizedExperiment 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.
swap.rownames	optionally named string or string vector. For SummarizedExperiment or SingleCellExperiment objects, its value(s) specifies the column name of rowData(object) to be used to identify features instead of rownames(object). When targeting multiple modalities (alternative experiments), names can be used to specify which level / alternative experiment (use 'main' for the top-level) individual values should be used for. See GeneTargeting for more specifics and examples.

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 Seurat, SingleCellExperiment, or SummarizedExperiment 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, adjustment = NULL, adj.fxn = NULL)
```

Arguments

`meta`	String, the name of the "metadata" slot to grab. OR "ident" to retireve the clustering of a Seurat `object`.
`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment object.
`adjustment`	A recognized string indicating whether numeric metadata should be used directly (default) versus 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] Ignored if the target metadata is not numeric.
`adj.fxn`	A function which takes a vector (of metadata values) and returns a vector of the same length. For example, `function(x) {log2(x)}` or `as.factor`

Details

Retrieves the values of a metadata slot from `object`, or the clustering slot if `meta = "ident"` and the `object` is a Seurat.

If `adjustment` or `adj.fxn` are provided, then these requested adjustments are applied to these values (`adjustment` first). Note: Alterations via `adjustment` are only applied when metadata is numeric, but `adj.fxn` alterations are applied to metadata of any type.

Lastly, outputs these values are named as the cells'/samples' names.

Value

A named vector.

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)

myRNA$numbers <- seq_len(ncol(myRNA))
meta("numbers", myRNA, adjustment = "z-score")
meta("numbers", myRNA, adj.fxn = as.factor)
meta("numbers", myRNA, adj.fxn = function(x) {log2(x)})
```

`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 Seurat, SingleCellExperiment, or SummarizedExperiment object.
`cells.use`	String vector of cells'/samples' names OR an integer vector specifying the indices of cells/samples 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.
`used.only`	TRUE by default, for target metadata that are factors, whether levels nonexistent in the target data should be ignored.

Value

String vector, the distinct values of a metadata slot (factor or not) among 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

```
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 dittoDimPlots for multiple features.*

Description

Generates dittoDimPlots for multiple features.

Usage

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

Arguments

`object` A Seurat, SingleCellExperiment, or SummarizedExperiment object.

`vars` c("var1","var2","var3",...). A vector of vars ('var' in regular `dittoDimPlot`) from which to generate the separate plots.

`ncol`, `nrow` Integer or NULL. How many columns or rows the plots should be arranged into.

`axes.labels.show` Logical. Whether axis labels should be shown. Subordinate to `xlab` and `ylab`.

`list.out` 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.

`OUT.List` Deprecated. Use `list.out`

`...`, `xlab`, `ylab`, `data.out`, `do.hover`, `legend.show` other parameters passed to `dittoDimPlot`.

Details

Given multiple 'var' parameters to `vars`, this function creates a `dittoDimPlot` for each one, with minor defaulting tweaks (see below).

By default, these dittoDimPlots are arranged into a grid. Alternatively, if `list.out` is set to `TRUE`, they are output as a list with each plot named as the `vars` being shown.

All parameters that can be adjusted in dittoDimPlot can be adjusted here, but the only input that will change between plots is `var`.

Value

A set of dittoDimPlots either arranged into a grid (default), or output as a list.

Slight tweaks to dittoDimPlot defaults

- axes labels are not shown by default to save space (control with `axes.labels.show` or `xlab` and `ylab`)
- legends are also not shown to save space (control with `legend.show`)

Author(s)

Daniel Bunis

See Also

`multi_dittoDimPlotVaryCells` for an alternate `dittoDimPlot` multi-plotter where the cells/samples are varied between plots.

`dittoDimPlot` for the base dittoDimPlot plotting function and details on all accepted inputs.

Examples

```
example(importDittoBulk, echo = FALSE)

multi_dittoDimPlot(myRNA, c("gene1", "gene2", "clustering"))

# Control grid shape with ncol / nrow
multi_dittoDimPlot(myRNA, c("gene1", "gene2", "clustering"),
    nrow = 1)

# Output as list instead
multi_dittoDimPlot(myRNA, c("gene1", "gene2", "clustering"),
    list.out = TRUE)
```

multi_dittoDimPlotVaryCells

Generates multiple dittoDimPlots, for a single feature, where each showing different cells

Description

Generates multiple dittoDimPlots, for a single feature, where each showing different cells

Usage

```
multi_dittoDimPlotVaryCells(
  object,
  var,
  vary.cells.meta,
  vary.cells.levels = metaLevels(vary.cells.meta, object),
  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,
  list.out = FALSE,
  OUT.List = NULL,
  ...,
  assay = .default_assay(object),
  slot = .default_slot(object),
  adjustment = NULL,
  min = NULL,
  max = NULL,
  color.panel = dittoColors(),
  colors = seq_along(color.panel),
  data.out = FALSE,
  do.hover = FALSE,
  swap.rownames = NULL
)
```

Arguments

`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment object.
`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, using colors, 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.

`vary.cells.meta`
: String name of a metadata that should be used for selecting which cells to show in each "VaryCells" `dittoDimPlot`.

`vary.cells.levels`
: The values/groupings of the `vary.cells.meta` metadata for which to generate a plot.

`show.titles`
: Logical which sets whether grouping-levels should be used as titles for the individual VaryCell plots. Default = TRUE.

`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` 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 inidividual VaryCell plots. 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 when `list.out` = `FALSE`.

`list.out`
: Logical which controls whether the list of plots should be returned as a list instead of as a single grid arrangement of the plots.

`OUT.List`
: Deprecated. Use `list.out`

`...`, `color.panel`, `colors`, `min`, `max`, `assay`, `slot`, `adjustment`, `data.out`, `do.hover`, `swap.rownames`
: additional parameters passed to `dittoDimPlot`.

  All parameters of `dittoDimPlot` can be utilized and adjusted except for `cells.use`, `main`, and `legend.show` which are handled with alternative methods here. 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.

Details

This function generates separate dittoDimPlots that show the same target data, but each for distinct cells.

How cells are separated into distinct plots is controlled with the `vary.cells.meta` parameter. Individual `dittoDimPlot`s are created for all levels of `var.cells.meta` groupings given to the `vary.cells.levels` input (default = all).

The function then appends a plot containing all cell/samples when `show.allcells.plot` = `TRUE`, with title of this plot controlled by `allcells.main`, as well as as single legend when `show.legend.single` = `TRUE`.

By default, these dittoDimPlots are output in a grid (default) with `ncol` columns and `nrow` rows, Alternatively, if `list.out` is set to `TRUE`, they are returned as a list. In the list, the VaryCell plots will be named by the levels of `vary.cells.meta` that they contain, and the optional allcells plot and single legend will be named "allcells" and "legend", respectively.

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 value across all plots.
- For discrete data, colors used in each plot are adjusted so that colors represent the same groupings across all plots.

Value

A set of dittoDimPlots either arranged into a grid (default), or output as a list.

Author(s)

Daniel Bunis

See Also

`multi_dittoDimPlot` for an alternate `dittoDimPlot` multi-plotter where `vars` are varied across plots rather than cells/samples

`dittoDimPlot` for the base dittoDimPlot plotting function and details on all accepted inputs.

Examples

```
example(importDittoBulk, echo = FALSE)

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

# Output as list instead
multi_dittoDimPlotVaryCells(myRNA, "gene1", vary.cells.meta = "clustering",
    list.out = TRUE)

# This function is also great for generating separate plots of each individual
#   grouping of a tsne/PCA/umap. This can be useful to check for dispersion
#   of groups that might otherwise be hidden behind other cells/samples.
#   The effect is similar to faceting, but: all distinct plots are treated
#   separately rather than being just a part of the whole, and with portrayal
#   of all cells/samples in an additional plot by default.
#
#   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 across clusters.
# (For an alternative method of viewing, and easily quantifying, how discrete
```

```
#   conditions of cells disperse across clusters, see '?dittoBarPlot')
multi_dittoDimPlotVaryCells(myRNA, "groups", vary.cells.meta = "clustering")
```

---

multi_dittoPlot *Generates dittoPlots for multiple features.*

Description

Generates dittoPlots for multiple features.

Usage

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

Arguments

`object`	A Seurat, SingleCellExperiment, or SummarizedExperiment object.
`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.
`ncol`, `nrow`	Integer or NULL. How many columns or rows the plots should be arranged into.
`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: For y-axis labels, gene vars will become "'var' expression". Equivalent to "var" for `main`. • 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.

list.out — 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.

OUT.List — Deprecated. Use list.out

..., xlab, data.out, do.hover, legend.show — other paramters passed along to dittoPlot.

Details

Given multiple 'var' parameters to vars, this function creates a dittoPlot for each one, with minor defaulting tweaks (see below).

By default, these dittoPlots are arranged into a grid. Alternatively, if list.out is set to TRUE, they are output as a list with each plot named as the vars being shown.

All parameters that can be adjusted in dittoPlot can be adjusted here, but the only input that will change between plots is the var.

Value

A set of dittoPlots either arranged into a grid (default), or output as a list.

Slight tweaks to dittoPlot defaults

- axes labels are not shown by default to save space (control with xlab and ylab)
- legends are also not shown to save space (control with legend.show)

Author(s)

Daniel Bunis

See Also

dittoPlot for the single plot version of this function and details on all accepted inputs.

dittoDotPlot and dittoPlotVarsAcrossGroups to show, in a single plot, per-group summaries of the values for multiple vars.

Examples

```
example(importDittoBulk, echo = FALSE)
genes <- getGenes(myRNA)[1:4]

multi_dittoPlot(myRNA,
    vars = c("gene1", "gene2", "gene3", "gene4"),
    group.by = "clustering")

#To make it output a grid that is 2x2, to add y-axis labels
# instead of titles, and to show legends...
multi_dittoPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "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

# Output as list instead
multi_dittoPlot(myRNA, c("gene1", "gene2", "gene3", "gene4"), "clustering",
    list.out = TRUE)
```

---

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

Description

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

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, 3, *5*, *23*
addPrcomp, *3*, 4
altExp, *85*
assay, *85*

Darken, 5, *81*
demux.calls.summary, 6, *9*, *93*
demux.SNP.summary, *7*, 7, *93*
demuxlet.example, 9
dittoBarPlot, 10, *23*, *39*, *81*
dittoBoxPlot, *64*
dittoBoxPlot *(*dittoPlot*)*, 56
dittoColors, 14, *19*, *50*, *76*, *81*
dittoDimHex, *23*, *80*
dittoDimHex *(*dittoHex*)*, 47
dittoDimPlot, *3–5*, 15, *55*, *80, 81*, *89*, *104, 105*, *107, 108*
dittoDotPlot, 25, *72*, *110*
dittoFreqPlot, *13*, 33
dittoHeatmap, 41, *81*
dittoHex, 46
dittoPlot, *8, 9*, *23*, *30*, *38*, 56, *72*, *81*, *110*
dittoPlotVarsAcrossGroups, *30*, *64*, 66, *110*
dittoRidgeJitter, *64*
dittoRidgeJitter *(*dittoPlot*)*, 56
dittoRidgePlot, *64*
dittoRidgePlot *(*dittoPlot*)*, 56
dittoScatterHex, *23*, *80*
dittoScatterHex *(*dittoHex*)*, 47
dittoScatterPlot, *23*, *55*, 73
dittoSeq, 81
dittoSeq-package *(*dittoSeq*)*, 81

facet_grid, *12*, *18*, *27*, *51*, *60*, *71*, *76*
facet_wrap, *12*, *18*, *27*, *35*, *51*, *60*, *71*, *76*

gene, *81*, 82, *87*, *99*
GeneTargeting, *19*, *28*, *42*, *51*, *53*, *59*, *68*, *76*, *82*, 83, *87*, *98*
geom_label_repel, *20*, *52*, *78*
geom_text_repel, *20*, *52*, *78*
geom_violin, *37*, *62*, *70*
getGenes, *23*, *55*, *80*, 87, *99*
getMetas, *23*, *55*, *80, 81*, 88, *100*, *102, 103*
getReductions, *23*, *55*, 89
ggrepel, *20*, *52*, *78*

Heatmap, *44, 45*, *81*

importDemux, *7*, *9*, 90
importDittoBulk, *3*, *5*, *23*, 94
isBulk, 97
isGene, *87*, 98
isMeta, *81*, *88*, 99, *102, 103*

Lighten, *81*, 101
linetype, *21*, *52*, *77*

max, *67*
mean, *67*
median, *67*
meta, *81*, *88*, *100*, 101, *103*
metaLevels, *12*, *29*, *34–36*, *45*, *60*, *69*, *102*, 103
multi_dittoDimPlot, 104, *108*
multi_dittoDimPlotVaryCells, *105*, 106
multi_dittoPlot, *30*, *64*, *72*, 109

pheatmap, *41*, *43–45*, *81*

rowData, *85*

scale_fill_gradient, *27*
scale_fill_gradient2, *27, 28*
setBulk, *98*, 111
setBulk,SingleCellExperiment-method *(*setBulk*)*, 111
Simulate, *81*, 112

SingleCellExperiment, *3–5*, *23*, *95*, *111*
slingshot, *21*, *52, 53*, *77*
stat_binline, *37*, *62*, *70*
SummarizedExperiment, *94*