Package ‘miaViz’

May 30, 2024

Title  Microbiome Analysis Plotting and Visualization
Version 1.12.0
Description The miaViz package implements functions to visualize
TreeSummarizedExperiment objects especially in the context of microbiome
analysis. Part of the mia family of R/Bioconductor packages.
biocViews Microbiome, Software, Visualization
License Artistic-2.0 | file LICENSE
Encoding UTF-8
LazyData false
Depends R (>= 4.0), SummarizedExperiment, TreeSummarizedExperiment,
mia (>= 0.99), ggplot2, ggraph (>= 2.0)
Imports methods, stats, S4Vectors, BiocGenerics, BiocParallel,
DelayedArray, scater, ggtree, ggnewscale, viridis, tidble,
tidytree, tidygraph, rlang, purrr, tidyr, dplyr, ape,
DirichletMultinomial, ggrepel, SingleCellExperiment
Suggests knitr, rmarkdown, BiocStyle, testthat, patchwork, vegan,
microbiomeDataSets, bluster
Roxygen list(markdown = TRUE)
RoxygenNote 7.3.1
VignetteBuilder knitr
git_url https://git.bioconductor.org/packages/miaViz
git_branch RELEASE_3_19
git_last_commit 945b0cc
git_last_commit_date 2024-04-30
Repository Bioconductor 3.19
Date/Publication 2024-05-29
Author Tuomas Borman [aut, cre] (<https://orcid.org/0000-0002-8563-8884>),
Felix G.M. Ernst [aut] (<https://orcid.org/0000-0001-5064-0928>),
Leo Lahti [aut] (<https://orcid.org/0000-0001-5537-637X>),
Basil Courbayre [ctb],
Giulio Benedetti [ctb] (<https://orcid.org/0000-0002-8732-7692>)
Maintainer Tuomas Borman <tuomas.v.borman@utu.fi>
miaViz-package

Contents

miaViz-package ......................................................... 2
mia-datasets .......................................................... 3
mia-plot-args ......................................................... 3
plotAbundance ......................................................... 6
plotAbundanceDensity ............................................. 8
plotCCA ............................................................... 11
plotColTile ........................................................... 14
plotDMN ............................................................... 15
plotGraph ............................................................. 16
plotPrevalence ......................................................... 20
plotSeries ............................................................. 23
plotTree ............................................................... 25
treeData ............................................................... 29

Index ........................................................................ 32

miaViz-package                                  miaViz - Microbiome Analysis Plotting and Visualization

Description

The scope of this package is the plotting and visualization of microbiome data. The main class for interfacing is the TreeSummarizedExperiment class.

Author(s)

Maintainer: Tuomas Borman <tuomas.v.borman@utu.fi> (ORCID)

Authors:

• Felix G.M. Ernst <felix.gm.ernst@outlook.com> (ORCID)
• Leo Lahti <leo.lahti@iki.fi> (ORCID)

Other contributors:

• Basil Courbayre [contributor]
• Giulio Benedetti <giulio.benedetti@utu.fi> (ORCID) [contributor]

See Also

mia class
Description

These example data objects were prepared to serve as examples. See the details for more information.

Usage

data(col_graph)

data(row_graph)

data(row_graph_order)

Format

An object of class tbl_graph (inherits from igraph) of length 26.
An object of class tbl_graph (inherits from igraph) of length 996.
An object of class tbl_graph (inherits from igraph) of length 110.

Details

For *_graph data:

1. “Jaccard” distances were calculated via calculateDistance(genus, FUN = vegan::vegdist, method = "jaccard", exprs_values = "relabundance"), either using transposed assay data or not to calculate distances for samples or features. NOTE: the function mia::calculateDistance is now deprecated.

2. “Jaccard” dissimilarities were converted to similarities and values above a threshold were used to construct a graph via graph.adjacency(mode = "lower", weighted = TRUE).

3. The igraph object was converted to tbl_graph via as_tbl_graph from the tidygraph package.

Description

To be able to fine tune plotting, several additional plotting arguments are available. These are described on this page.
Tree plotting

- `line_alpha`: Numeric scalar in [0, 1], specifying the transparency of the tree edges. Defaults to 1.
- `line_width`: Numeric scalar, specifying the default width of an edge. Defaults to NULL to use default of the ggtree package.
- `line_width_range`: Two numeric values, the range for plotting dynamic edge widths in. Defaults to c(0.5, 3).
- `point_alpha`: Numeric scalar in [0, 1], specifying the transparency of the tips. Defaults to 1.
- `point_size`: Numeric scalar, specifying the default size of tips. Defaults to 2.
- `point_size_range`: Two numeric values, the range for plotting dynamic tip sizes in. Defaults to c(1, 4).
- `label_font_size`: Numeric scalar, font size for the tip and node labels. Default to 3.
- `highlight_font_size`: Numeric scalar, font size for the highlight labels. Default to 3.

Graph plotting

- `line_alpha`: Numeric scalar in [0, 1], specifying the transparency of the tree edges. Defaults to 1.
- `line_width`: Numeric scalar, specifying the default width of an edge. Defaults to NULL to use default of the ggraph package.
- `line_width_range`: Two numeric values, the range for plotting dynamic edge widths in. Defaults to c(0.5, 3).
- `point_alpha`: Numeric scalar in [0, 1], specifying the transparency of the tips. Defaults to 1.
- `point_size`: Numeric scalar, specifying the default size of tips. Defaults to 2.
- `point_size_range`: Two numeric values, the range for plotting dynamic tip sizes in. Defaults to c(1, 4).

Abundance plotting

- `flipped`: Logical scalar. Should the plot be flipped? Defaults to FALSE.
- `add_legend`: Logical scalar. Should legends be plotted? Defaults to TRUE.
- `add_x_text`: Logical scalar. Should x tick labels be plotted? Defaults to FALSE.
- `add_border`: Logical scalar. Should border of bars be plotted? Defaults to FALSE.
- `bar_alpha`: Numeric scalar in [0, 1], specifying the transparency of the bars. Defaults to 1.
- `point_alpha`: Numeric scalar in [0, 1], specifying the transparency of the points. Defaults to 1.
- `point_size`: Numeric scalar, specifying the default size of points. Defaults to 2.

Abundance density plotting

- `add_legend`: Logical scalar. Should legends be plotted? Defaults to TRUE.
- `point_shape`: Numeric scalar setting the shape of points. Defaults to 21.
- `point_colour`: Character scalar, specifying the default colour of points. Defaults to 2.
point_size: Numeric scalar, specifying the default size of points. Defaults to 2.
point_alpha: Numeric scalar in [0, 1], specifying the transparency of the points. Defaults to 1.
flipped: Logical scalar. Should the plot be flipped? Defaults to FALSE.
scales_free: Logical scalar. Should scales = "free" be set for faceted plots? Defaults to TRUE.
angle_x_text: Logical scalar. Should x tick labels be plotted? Defaults to FALSE.

Prevalence plotting

flipped: Logical scalar, specifying whether the plot should be flipped. Defaults to FALSE.
add_legend: Logical scalar. Should legends be plotted? Defaults to TRUE.
point_alpha: Numeric scalar in [0, 1], specifying the transparency of the tips. Defaults to 1.
point_size: Numeric scalar, specifying the default size of tips. Defaults to 2.
line_alpha: Numeric scalar in [0, 1], specifying the transparency of the tree edges. Defaults to 1.
line_type: Numeric scalar, specifying the default line type. Defaults to NULL to use default of the ggplot2 package.
line_size: Numeric scalar, specifying the default width of a line. Defaults to NULL to use default of the ggplot2 package.

Series plotting

add_legend: Logical scalar. Should legends be plotted? Defaults to TRUE.
line_alpha: Numeric scalar in [0, 1], specifying the transparency of the tree edges. Defaults to 1.
line_type: Numeric scalar, specifying the default line type. Defaults to NULL to use default of the ggplot2 package.
line_width: Numeric scalar, specifying the default width of a line. Defaults to NULL to use default of the ggplot2 package.
line_width_range: Two numeric values, the range for plotting dynamic line widths in. Defaults to c(0.5, 3).
ribbon_alpha: Numeric scalar in [0, 1], specifying the transparency of the ribbon. Defaults to 0.3.

Tile plotting

add_legend: Logical scalar. Should legends be plotted? Defaults to TRUE.
rect_alpha: Numeric scalar in [0, 1], specifying the transparency of the areas. Defaults to 1.
rect_colour: Character scalar, specifying the colour to use for colouring the borders of the areas. Defaults to "black".
na.value: Character scalar, specifying the colour to use for NA values. Defaults to "grey80".
plotAbundance  

Plotting abundance data

Description

plotAbundance plots the abundance on a selected taxonomic rank. Since this probably makes sense only for relative abundance data, the assay used by default is expected to be in the slot ‘relabundance’. If only ‘counts’ is present, the relative abundance is computed.

Usage

plotAbundance(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotAbundance(
x,  
rank = taxonomyRanks(x)[1],  
features = NULL,  
order_rank_by = c("name", "abund", "revabund"),  
order_sample_by = NULL,  
decreasing = TRUE,  
use_relative = TRUE,  
layout = c("bar", "point"),  
one_facet = TRUE,  
ncol = 2,  
scales = "fixed",  
assay.type = assay_name,  
assay_name = "counts",  
...  
)

Arguments

x  
a SummarizedExperiment object.

...  
additional parameters for plotting. See mia-plot-args for more details i.e. call help("mia-plot-args")

rank  
a single character value defining the taxonomic rank to use. Must be a value of taxonomyRanks(x).

features  
a single character value defining a column from colData to be plotted below the abundance plot. Continuous numeric values will be plotted as point, whereas factors and character will be plotted as colour-code bar. (default: features = NULL)

order_rank_by  
How to order abundance value: By name (“name”) for sorting the taxonomic labels alphabetically, by abundance (“abund”) to sort by abundance values or by a reverse order of abundance values (“revabund”).
plotAbundance

order_sample_by
A single character value from the chosen rank of abundance data or from colData to select values to order the abundance plot by. (default: order_sample_by = NULL)

decreasing
TRUE or FALSE: If the order_sample_by is defined and the values are numeric, should the values used to order in decreasing or increasing fashion? (default: decreasing = FALSE)

use_relative
TRUE or FALSE: Should the relative values be calculated? (default: use_relative = TRUE)

layout
Either “bar” or “point”.

one_facet
Should the plot be returned in on facet or split into different facet, one facet per different value detect in rank. If features or order_sample_by is not NULL, this setting will be disregarded.

ncol, scales
if one_facet = FALSE, ncol defines many columns should be for plotting the different facets and scales is used to define the behavior of the scales of each facet. Both values are passed onto facet_wrap.

assay.type
a character value defining which assay data to use. (default: assay.type = “relabundance”)

assay_name
a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)

Details
Subsetting to rows of interested and ordering of those is expected to be done outside of this function, e.g. x[1:2,]. This will plot data of all features present.

Value
a ggplot object or list of two ggplot objects, if features are added to the plot.

Examples

data(GlobalPatterns, package="mia")
se <- GlobalPatterns

## Plotting abundance using the first taxonomic rank as default
plotAbundance(se, assay.type="counts")

## Using “Phylum” as rank
plotAbundance(se, assay.type="counts", rank = "Phylum", add_legend = FALSE)

## If rank is set to NULL plotAbundance behaves like plotExpression
plotAbundance(se, assay.type="counts", rank = NULL,
features = head(rownames(se)))

## A feature from colData or taxon from chosen rank can be used for ordering samples.
plotAbundance(se, assay.type="counts", rank = "Phylum",
order_sample_by = "Bacteroidetes")
plotAbundanceDensity  

**Plot abundance density**

**Description**

This function plots abundance of the most abundant taxa.

**Usage**

`plotAbundanceDensity(object, ...)`

## S4 method for signature 'SummarizedExperiment'

plotAbundanceDensity(object, ...)

```r
## S4 method for signature 'SummarizedExperiment'
plotAbundanceDensity(
## Features from colData can be plotted together with abundance plot.  
## Returned object is a list that includes two plot; other visualizes abundance 
## other features.
plot <- plotAbundance(se, assay.type = "counts", rank = "Phylum",  
  features = "SampleType")

# These two plots can be combined with wrap_plots function from patchwork package
library(patchwork)
wrap_plots(plot, ncol = 1)

## Same plot as above but showing sample IDs as labels for the x axis on the top plot
plot[[1]] <- plotAbundance(se, assay.type = "counts", rank = "Phylum",  
  features = "SampleType", add_legend = FALSE,  
  add_x_text = TRUE)[[1]] +  
  theme(axis.text.x = element_text(angle = 90))

wrap_plots(plot, ncol = 1, heights = c(0.8,0.2))

## Compositional barplot with top 5 taxa and samples sorted by "Bacteroidetes"

# Getting top taxa on a Phylum level
se <- transformAssay(se, method="relabundance")
se_phylum <- agglomerateByRank(se, rank ="Phylum", onRankOnly=TRUE)
top_taxa <- getTopFeatures(se_phylum,top = 5, assay.type = "relabundance")

# Renaming the "Phylum" rank to keep only top taxa and the rest to "Other"
phylum_renamed <- lapply(rowData(se)$Phylum,  
  function(x){if (x %in% top_taxa) {x} else {"Other"}})
rowData(se)$Phylum <- as.character(phylum_renamed)

# Compositional barplot
plotAbundance(se, assay.type="relabundance", rank = "Phylum",  
  order_rank_by="abund", order_sample_by = "Bacteroidetes")
```
plotAbundanceDensity

```r
object,
layout = c("jitter", "density", "point"),
assay.type = assay_name,
assay_name = "counts",
n = min(nrow(object), 25L),
colour_by = NULL,
shape_by = NULL,
size_by = NULL,
order_descending = TRUE,
...
)
```

Arguments

- **object**: a `SummarizedExperiment` object.
- **...**: additional parameters for plotting.
  - `xlab`: a single character value for selecting the x-axis label. (default: `xlab = assay.type`)
  - `ylab`: a single character value for selecting the y-axis label. `ylab` is disabled when `layout = "density"`. (default: `ylab = "Taxa"`)
  - `point_alpha`: a numeric value from range 0 to 1 selecting the transparency of colour in `jitter` and `point` plot. (default: `point_alpha = 0.6`)
  - `point_shape`: a positive integer value selecting the shape of point in `jitter` and `point` plot. (default: `point_shape = 21`)
  - `point_size`: a positive numeric value selecting the size of point in `jitter` and `point` plot. (default: `point_size = 2`)
  - `add_legend`: a boolean value selecting if legend is added. (default: `add_legend = TRUE`)
  - `flipped`: a boolean value selecting if the orientation of plot is changed so that x-axis and y-axis are swapped. (default: `flipped = FALSE`)
  - `add_x_text`: a boolean value selecting if text that represents values is included in x-axis. (default: `add_x_text = TRUE`)

See `mia-plot-args` for more details i.e. call `help("mia-plot-args")`

- **layout**: a single character value for selecting the layout of the plot. There are three different options: `jitter`, `density`, and `point` plot. (default: `layout = "jitter"`)
- **assay.type**: a single character value for selecting the `assay` to be plotted. (default: `assay.type = "counts"`)
- **assay_name**: a single character value for specifying which assay to use for calculation. (Please use `assay.type` instead. At some point `assay_name` will be disabled.)
- **n**: a positive integer specifying the number of the most abundant taxa to show. (default: `n = min(nrow(object), 25L)`)
- **colour_by**: a single character value defining a column from `colData`, that is used to color plot. Must be a value of `colData()` function. (default: `colour_by = NULL`)
- **shape_by**: a single character value defining a column from `colData`, that is used to group observations to different point shape groups. Must be a value of `colData()`
function. shape_by is disabled when layout = "density". (default: shape_by = NULL)

size_by a single character value defining a column from colData, that is used to group observations to different point size groups. Must be a value of colData() function. size_by is disabled when layout = "density". (default: size_by = NULL)

order_descending TRUE, FALSE or NA: Should the results be ordered in a descending order? If NA is given the order as found in object for the n most abundant taxa is used. (default: order_descending = TRUE)

Details

This function plots abundance of the most abundant taxa. Abundance can be plotted as a jitter plot, a density plot, or a point plot. By default, x-axis represents abundance and y-axis taxa. In a jitter and point plot, each point represents abundance of individual taxa in individual sample. Most common abundances are shown as a higher density.

A density plot can be seen as a smoothened bar plot. It visualized distribution of abundances where peaks represent most common abundances.

Value

A ggplot2 object

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

Examples

tse <- microbiomeDataSets::atlas1006()

# Plots the abundances of 25 most abundant taxa. Jitter plot is the default option.
plotAbundanceDensity(tse, assay.type = "counts")

# Counts relative abundances
tse <- transformAssay(tse, method = "relabundance")

# Plots the relative abundance of 10 most abundant taxa.
# "nationality" information is used to color the points. X-axis is log-scaled.
plotAbundanceDensity(tse, layout = "jitter", assay.type = "relabundance", n = 10, colour_by = "nationality") +
  scale_x_log10()

# Plots the relative abundance of 10 most abundant taxa as a density plot.
# X-axis is log-scaled
plotAbundanceDensity(tse, layout = "density", assay.type = "relabundance", n = 10 ) +
  scale_x_log10()
# Plots the relative abundance of 10 most abundant taxa as a point plot.
# Point shape is changed from default (21) to 41.
plotAbundanceDensity(tse, layout = "point", assay.type = "relabundance", n = 10,
    point_shape = 41)

# Plots the relative abundance of 10 most abundant taxa as a point plot.
# In addition to colour, groups can be visualized by size and shape in point plots,
# and adjusted for point size
plotAbundanceDensity(tse, layout = "point", assay.type = "relabundance", n = 10,
    shape_by = "sex", size_by = "time", point_size=1)

# Ordering via order_descending
plotAbundanceDensity(tse, assay.type = "relabundance",
    order_descending = FALSE)

# for custom ordering set order_descending = NA and order the input object
# to your wishes
plotAbundanceDensity(tse, assay.type = "relabundance",
    order_descending = NA)

---

plotCCA

**Plot RDA or CCA object**

**Description**

plotRDA and plotCCA create an RDA/CCA plot starting from the output of CCA and RDA functions, two common methods for supervised ordination of microbiome data.

**Usage**

plotCCA(object, ...)

## S4 method for signature 'SingleCellExperiment'
plotCCA(object, dimred, ...)

## S4 method for signature 'matrix'
plotCCA(object, ...)

plotRDA(object, ...)

## S4 method for signature 'SingleCellExperiment'
plotRDA(
    object,
    dimred,
    add.ellipse = TRUE,
    ellipse.alpha = 0.2,
    ellipse.linewidth = 0.1,
    ellipse.linetype = 1,
```r
vec.size = 0.5,
vec.color = vec.colour,
vec.colour = "black",
vec.linetype = 1,
arrow.size = 0.25,
label.color = label.colour,
label.colour = "black",
label.size = 4,
vec.text = TRUE,
repel.labels = TRUE,
sep.group = "–",
repl.underscore = " ",
add.significance = TRUE,
add.expl.var = TRUE,
add.vectors = TRUE,
parse.labels = TRUE,

## S4 method for signature 'matrix'
plotRDA(object, ...)
```

### Arguments

- `object` a `TreeSummarizedExperiment` or a matrix of weights. The latter is returned as output from `calculateRDA`.
- `...` additional parameters for plotting, inherited from `plotReducedDim`, `geom_label` and `geom_label_repel`.
- `dimred` A string or integer scalar indicating the reduced dimension to plot. This is the output of `runRDA` and resides in `reducedDim(tse, dimred)`.
- `add.ellipse` One of `c(TRUE, FALSE, "fill", "colour", "color")`, indicating whether ellipses should be present, absent, filled or colored. (default: `ellipse.fill = TRUE`)
- `ellipse.alpha` Number between 0 and 1 to adjust the opacity of ellipses. (default: `ellipse.alpha = 0.2`)
- `ellipse.linewidth` Number specifying the size of ellipses. (default: `ellipse.linewidth = 0.1`)
- `ellipse.linetype` Discrete number specifying the style of ellipses. (default: `ellipse.linetype = 1`)
- `vec.size` Number specifying the size of vectors. (default: `vec.size = 0.5`)
- `vec.color` Alias for `vec.colour`.
- `vec.colour` String specifying the colour of vectors. (default: `vec.color = "black"`)
- `vec.linetype` Discrete number specifying the style of vector lines. (default: `vec.linetype = 1`)
- `arrow.size` Number specifying the size of arrows. (defaults: `arrow.size = 0.25`)
label.color  Alias for label.colour.
label.colour  String specifying the colour of text and labels. (default: label.color = "black")
label.size   Number specifying the size of text and labels. (default: label.size = 4)
vec.text     TRUE or FALSE, should text instead of labels be used to label vectors. (default: vec.text = TRUE)
repel.labels TRUE or FALSE, should labels be repelled. (default: repel.labels = TRUE)
sep.group    String specifying the separator used in the labels. (default: sep.group = "\U2012")
repl.underscore  String used to replace underscores in the labels. (default: repl.underscore = " ")
add.significance  TRUE or FALSE, should explained variance and p-value appear in the labels. (default: add.significance = TRUE)
add.expl,var   TRUE or FALSE, should explained variance appear on the coordinate axes. (default: add.expl.var = TRUE)
add.vectors    TRUE or FALSE, should vectors appear in the plot. (default: add.vectors = TRUE)
parse.labels   TRUE or FALSE, should labels be parsed. (default: parse.labels = TRUE)

Details

plotRDA and plotCCA create an RDA/CCA plot starting from the output of CCA and RDA functions, two common methods for supervised ordination of microbiome data. Either a TreeSummarizedExperiment or a matrix object is supported as input. When the input is a TreeSummarizedExperiment, this should contain the output of runRDA in the reducedDim slot and the argument dimred needs to be defined. When the input is a matrix, this should be returned as output from calculateRDA. However, the first method is recommended because it provides the option to adjust aesthetics to the colData variables through the arguments inherited from plotReducedDim.

Value

A ggplot2 object

Examples

# Load dataset
library(miaViz)
data("enterotype", package = "mia")
tse <- enterotype

# Run RDA and store results into TreeSE
tse <- runRDA(tse,
  formula = assay ~ ClinicalStatus + Gender + Age,
  FUN = vegan::vegdist,
  distance = "bray",
  na.action = na.exclude)
# Create RDA plot coloured by variable
plotRDA(tse, "RDA",
       colour_by = "ClinicalStatus")

# Create RDA plot with empty ellipses
plotRDA(tse, "RDA",
       colour_by = "ClinicalStatus",
       add.ellipse = "colour")

# Create RDA plot with text encased in labels
plotRDA(tse, "RDA",
       colour_by = "ClinicalStatus",
       vec.text = FALSE)

# Create RDA plot without repelling text
plotRDA(tse, "RDA",
       colour_by = "ClinicalStatus",
       repel.labels = FALSE)

# Create RDA plot without vectors
plotRDA(tse, "RDA",
       colour_by = "ClinicalStatus",
       add.vectors = FALSE)

# Calculate RDA as a separate object
rda_mat <- calculateRDA(tse,
                        formula = assay ~ ClinicalStatus + Gender + Age,
                        FUN = vegan::vegdist,
                        distance = "bray",
                        na.action = na.exclude)

# Create RDA plot from RDA matrix
plotRDA(rda_mat)

---

plotColTile

*Plot factor data as tiles*

**Description**

Relative relations of two grouping can be visualized by plotting tiles with relative sizes. `plotColTile` and `plotRowTile` can be used for this.

**Usage**

```r
plotColTile(object, x, y, ...)
```

```r
plotRowTile(object, x, y, ...)
```

```
# S4 method for signature 'SummarizedExperiment'
plotColTile(object, x, y, ...)
```
## Arguments

object  
a SummarizedExperiment object.

x  
String specifying the column-level metadata field to show on the x-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo or ?retrieveCellInfo. Must result in a returned character or factor vector.

y  
String specifying the column-level metadata to show on the y-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo or ?retrieveCellInfo. Must result in a returned character or factor vector.

...  
additional arguments for plotting. See mia-plot-args for more details i.e. call help("mia-plot-args")

## Value

A ggplot2 object or plotly object, if more than one prevalences was defined.

## Examples

data(GlobalPatterns)
se <- GlobalPatterns
plotColTile(se,"SampleType","Primer")

## Description

To plot DMN fits generated with mia use plotDMNFit.

## Usage

plotDMNFit(x, name = "DMN", type = c("laplace", "AIC", "BIC"), ...)

## Arguments

x  
a SummarizedExperiment object contain the DMN data in metadata.

name  
the name to store the result in metadata

type  
the type of measure for access the goodness of fit. One of ‘laplace’, ‘AIC’ or ‘BIC’.

...  
optional arguments not used.
plotDMNFit returns a ggplot2 plot.

See Also

calculateDMN

Examples

library(mia)
library(bluster)

# Get dataset
data("peerj13075", package = "mia")
tse <- peerj13075

# Cluster the samples
tse <- cluster(tse, DmmParam(k = 1:4), name = "DMM", full = TRUE)

# Plot the fit
plotDMNFit(tse, name = "DMM", type = "laplace")
colour_by = NULL,
shape_by = NULL,
size_by = NULL,
by_exprs_values = "counts",
other_fields = list(),
...
)

## S4 method for signature 'SummarizedExperiment,missing'
plotColGraph(x, y, name = "graph", ...)

## S4 method for signature 'ANY,SummarizedExperiment'
plotRowGraph(
  x,
  y,
  show_label = FALSE,
  add_legend = TRUE,
  layout = "kk",
  edge_type = c("fan", "link", "arc", "parallel"),
  edge_colour_by = NULL,
  edge_width_by = NULL,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = "counts",
  other_fields = list(),
  ...
)

## S4 method for signature 'SummarizedExperiment,missing'
plotRowGraph(x, y, name = "graph", ...)

Arguments

x, y a graph object and a SummarizedExperiment object or just a SummarizedExperiment. For the latter object a graph object must be stored in metadata(x)$name.

... additional arguments for plotting. See mia-plot-args for more details i.e. call help("mia-plot-args")

show_label logical (scalar), integer or character vector. If a logical scalar is given, should tip labels be plotted or if a logical vector is provided, which labels should be shown? If an integer or character vector is provided, it will be converted to a logical vector. The integer values must be in the range of 1 and number of nodes, whereas the values of a character vector must match values of a label or name column in the node data. In case of a character vector only values corresponding to actual labels will be plotted and if no labels are provided no labels will be shown. (default: show_label = FALSE)

add_legend logical scalar. Should legends be plotted? (default: add_legend = TRUE)

layout layout for the plotted graph. See ggplot for details. (default: layout = "kk")
edge_type: type of edge plotted on the graph. See `geom_edge_fan` for details and other available geoms. (default: `edge_type = "fan"`)

edge_colour_by: Specification of a edge metadata field to use for setting colours of the edges.

data_width_by: Specification of a edge metadata field to use for setting width of the edges.

colour_by: Specification of a column metadata field or a feature to colour graph nodes by,
see the by argument in `?retrieveCellInfo` for possible values.

shape_by: Specification of a column metadata field or a feature to shape graph nodes by,
see the by argument in `?retrieveCellInfo` for possible values.

size_by: Specification of a column metadata field or a feature to size graph nodes by, see
the by argument in `?retrieveCellInfo` for possible values.

by_exprs_values: A string or integer scalar specifying which assay to obtain expression values
from, for use in point aesthetics - see the exprs_values argument in `?retrieveCellInfo`.

other_fields: Additional fields to include in the node information without plotting them.

colour: If `x` is a `SummarizedExperiment` the key for subsetting the metadata(x) to a
graph object.

Details: Internally `tidygraph` and `ggraph` are used. Therefore, all graph types which can be converted by
tidygraph::as_tbl_graph can be used.

Value: a `ggtree` plot

Examples:

```
# data setup
library(mia)
data(GlobalPatterns)
data(col_graph)
data(row_graph)
data(row_graph_order)
metadata(GlobalPatterns)$col_graph <- col_graph

metadata(genus)$row_graph <- row_graph
metadata(order)$row_graph <- row_graph_order

genus <- agglomerateByRank(GlobalPatterns,"Genus",na.rm=TRUE)
metadata(genus)$row_graph <- row_graph
order <- agglomerateByRank(genus,"Order",na.rm=TRUE)
metadata(order)$row_graph <- row_graph_order

# plot a graph independently
plotColGraph(col_graph,
genus,
colour_by = "SampleType",
edge_colour_by = "weight",
edge_width_by = "weight",
show_label = TRUE)
```
# plot the graph stored in the object
plotColGraph(genus,
  name = "col_graph",
  colour_by = "SampleType",
  edge_colour_by = "weight",
  edge_width_by = "weight")

# plot a graph independently
plotRowGraph(row_graph,
  genus,
  colour_by = "Kingdom",
  edge_colour_by = "weight",
  edge_width_by = "weight")

# plot the graph stored in the object
plotRowGraph(genus,
  name = "row_graph",
  colour_by = "Phylum",
  edge_colour_by = "weight",
  edge_width_by = "weight")

# plot a graph independently
plotRowGraph(row_graph_order,
  order,
  colour_by = "Kingdom",
  edge_colour_by = "weight",
  edge_width_by = "weight")

# plot the graph stored in the object and include some labels
plotRowGraph(order,
  name = "row_graph",
  colour_by = "Phylum",
  edge_colour_by = "weight",
  edge_width_by = "weight",
  show_label = c("Sulfolobales", "Spirochaetales", 
                 "Verrucomicrobiales"))

# labels can also be included via selecting specific rownames of x/y
plotRowGraph(order,
  name = "row_graph",
  colour_by = "Phylum",
  edge_colour_by = "weight",
  edge_width_by = "weight",
  show_label = c(1,10,50))

# labels can also be included via a logical vector, which has the same length
# as nodes are present
label_select <- rep(FALSE,nrow(order))
label_select[c(1,10,50)] <- TRUE
plotRowGraph(order,
  name = "row_graph",
\begin{verbatim}
colour_by = "Phylum",
edge_colour_by = "weight",
edge_width_by = "weight",
show_label = label_select)
\end{verbatim}

---

**plotPrevalence**

**Plot prevalence information**

**Description**

plotPrevalence and plotFeaturePrevalence visualize prevalence information.

**Usage**

```r
plotPrevalence(x, ...)
```

\[
## S4 method for signature 'SummarizedExperiment'
plotPrevalence(
x,
detections = c(0.01, 0.1, 1, 2, 5, 10, 20)/100,
prevalences = seq(0.1, 1, 0.1),
assay.type = assay_name,
assay_name = "counts",
as_relative = TRUE,
rank = NULL,
BPPARAM = BiocParallel::SerialParam(),
...)
```

```r
plotPrevalentAbundance(x, ...)
```

\[
## S4 method for signature 'SummarizedExperiment'
plotPrevalentAbundance(
x,
rank = taxonomyRanks(x)[1L],
assay.type = assay_name,
assay_name = "counts",
as_relative = TRUE,
colour_by = NULL,
size_by = NULL,
shape_by = NULL,
label = NULL,
facet_by = NULL,
...)
\]
plotFeaturePrevalence(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotFeaturePrevalence(  
  x,  
  rank = taxonomyRanks(x)[1L],  
  assay.type = assay_name,  
  assay_name = "counts",  
  detections = NULL,  
  ndetections = 20,  
  as_relative = TRUE,  
  min_prevalence = 0,  
  BPPARAM = BiocParallel::SerialParam(),  
  ...  
)

plotTaxaPrevalence(x, ...)

## S4 method for signature 'ANY'
plotTaxaPrevalence(x, ...)

Arguments

x a SummarizedExperiment object.
detections Detection thresholds for absence/presence. Either an absolute value compared directly to the values of x or a relative value between 0 and 1, if as_relative = TRUE.
prevalences Prevalence thresholds (in 0 to 1). The required prevalence is strictly greater by default. To include the limit, set include_lowest to TRUE.
assay.type a character value defining which assay data to use. (default: assay.type = "relabundance")
assay_name a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
as_relative logical scalar: Should the detection threshold be applied on compositional (relative) abundances? Passed onto getPrevalence. (default: TRUE)
rank,... additional arguments
  • If !is.null(rank) matching arguments are passed on to agglomerateByRank. See ?agglomerateByRank for more details.
  • additional arguments for plotting. See mia-plot-args for more details i.e. call help("mia-plot-args")
BPPARAM A BiocParallelParam object specifying whether the UniFrac calculation should be parallelized.
colour_by Specification of a feature to colour points by, see the by argument in ?retrieveFeatureInfo for possible values. Only used with layout = "point".
size_by Specification of a feature to size points by, see the by argument in ?retrieveFeatureInfo for possible values. Only used with layout = "point".
**plotPrevalence**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shape_by</code></td>
<td>Specification of a feature to shape points by, see the by argument in <code>?retrieveFeatureInfo</code> for possible values. Only used with layout = &quot;point&quot;.</td>
</tr>
<tr>
<td><code>label</code></td>
<td>a logical, character or integer vector for selecting labels from the row-names of <code>x</code>. If <code>rank</code> is not NULL the rownames might change. (default: <code>label</code> = <code>NULL</code>)</td>
</tr>
<tr>
<td><code>facet_by</code></td>
<td>Taxonomic rank to facet the plot by. Value must be of <code>taxonomyRanks(x)</code> Argument can only be used in function <code>plotPrevalentAbundance</code></td>
</tr>
<tr>
<td><code>ndetections</code></td>
<td>If <code>detections</code> is <code>NULL</code>, a number of breaks are calculated automatically. <code>as_relative</code> is then also regarded as <code>TRUE</code>.</td>
</tr>
<tr>
<td><code>min_prevalence</code></td>
<td>a single numeric value to apply as a threshold for plotting. The threshold is applied per row and column. (default: <code>min_prevalence</code> = <code>0</code>)</td>
</tr>
</tbody>
</table>

**Details**

Whereas `plotPrevalence` produces a line plot, `plotFeaturePrevalence` returns a heatmap. Agglomeration on different taxonomic levels is available through the `rank` argument.

To exclude certain taxa, preprocess `x` to your liking, for example with subsetting via `getPrevalentTaxa` or `agglomerateByPrevalence`.  

**Value**

A ggplot2 object or plotly object, if more than one prevalences was defined.

**See Also**

`getPrevalence`, `agglomerateByPrevalence`, `agglomerateByRank`

**Examples**

```r
data(GlobalPatterns, package = "mia")

# plotting N of prevalence exceeding taxa on the Phylum level
plotPrevalence(GlobalPatterns, rank = "Phylum")
plotPrevalence(GlobalPatterns, rank = "Phylum") + scale_x_log10()

# plotting prevalence per taxa for different detection thresholds as heatmap
plotFeaturePrevalence(GlobalPatterns, rank = "Phylum")

# by default a continuous scale is used for different detection levels, # but this can be adjusted
plotFeaturePrevalence(GlobalPatterns, rank = "Phylum",
  detections = c(0, 0.001, 0.01, 0.1, 0.2))

# point layout for plotFeaturePrevalence can be used to visualize by additional # information
plotPrevalentAbundance(GlobalPatterns, rank = "Family",
  colour_by = "Phylum") +
  scale_x_log10()
```
plotSeries

# When using function plotPrevalentAbundance, it is possible to create facets
# with 'facet_by'.
plotPrevalentAbundance(GlobalPatterns, rank = "Family",
        colour_by = "Phylum", facet_by = "Kingdom") +
        scale_x_log10()

---

plotSeries  Plot Series

Description

This function plots series data.

Usage

plotSeries(
    object,
    x,
    y = NULL,
    rank = NULL,
    colour_by = NULL,
    size_by = NULL,
    linetype_by = NULL,
    assay.type = assay_name,
    assay_name = "counts",
    ...
)

## S4 method for signature 'SummarizedExperiment'
plotSeries(
    object,
    x,
    y = NULL,
    rank = NULL,
    colour_by = NULL,
    size_by = NULL,
    linetype_by = NULL,
    assay.type = assay_name,
    assay_name = "counts",
    ...
)

Arguments

object  a SummarizedExperiment object.

x  a single character value for selecting the column from ColData that will specify values of x-axis.
a single character value for selecting the taxa from `rownames`. This parameter specifies taxa whose abundances will be plotted.

- **rank**: a single character value defining a taxonomic rank, that is used to agglomerate the data. Must be a value of `taxonomicRanks()` function.

- **colour_by**: a single character value defining a taxonomic rank, that is used to color plot. Must be a value of `taxonomicRanks()` function.

- **size_by**: a single character value defining a taxonomic rank, that is used to divide taxa to different line size types. Must be a value of `taxonomicRanks()` function.

- **linetype_by**: a single character value defining a taxonomic rank, that is used to divide taxa to different line types. Must be a value of `taxonomicRanks()` function.

- **assay.type**: a single character value for selecting the assay to be plotted. (default: `assay.type = "counts"`)

- **assay_name**: a single character value for specifying which assay to use for calculation. (Please use `assay.type` instead. At some point `assay.name` will be disabled.)

- **...**: additional parameters for plotting. See `mia-plot-args` for more details i.e. call `help("mia-plot-args")`

### Details
This function creates series plot, where x-axis includes e.g. time points, and y-axis abundances of selected taxa.

### Value
A `ggplot2` object

### Author(s)
Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

### Examples
```r
## Not run:
library(mia)
# Load data from miaTime package
library("miaTime")
data(SilvermanAGutData)
object <- SilvermanAGutData

# Plots 2 most abundant taxa, which are colored by their family
plotSeries(object,
  x = "DAY_ORDER",
  y = getTopFeatures(object, 2),
  colour_by = "Family")

# Counts relative abundances
object <- transformAssay(object, method = "relabundance")

# Selects taxa
```
taxa <- c("seq_1", "seq_2", "seq_3", "seq_4", "seq_5")

# Plots relative abundances of phylums
plotSeries(object[ taxa, ],
    x = "DAY_ORDER",
    colour_by = "Family",
    linetype_by = "Phylum",
    assay.type = "relabundance")

# In addition to 'colour_by' and 'linetype_by', 'size_by' can also be used to group taxa.
plotSeries(object,
    x = "DAY_ORDER",
    y = getTopFeatures(object, 5),
    colour_by = "Family",
    size_by = "Phylum",
    assay.type = "counts")

## End(Not run)

---

**plotTree**

Plotting tree information enriched with information

**Description**

Based on the stored data in a `TreeSummarizedExperiment` a tree can be plotted. From the rowData, the assays as well as the colData information can be taken for enriching the tree plots with additional information.

**Usage**

```r
plotRowTree(object, ...)
plotColTree(object, ...)
```

```
## S4 method for signature 'TreeSummarizedExperiment'
plotColTree(
    object,
    tree_name = "phylo",
    relabel_tree = FALSE,
    order_tree = FALSE,
    remove_levels = FALSE,
    show_label = FALSE,
    show_highlights = FALSE,
    show_highlight_label = FALSE,
    abbr_label = FALSE,
    add_legend = TRUE,
    layout = "circular",
    edge_colour_by = NULL,
)```
```r
plotTree

edge_size_by = NULL,
tip_colour_by = NULL,
tip_shape_by = NULL,
tip_size_by = NULL,
node_colour_by = NULL,
node_shape_by = NULL,
node_size_by = NULL,
colour_highlights_by = NULL,
by_exprs_values = "counts",
other_fields = list(),
...

## S4 method for signature 'TreeSummarizedExperiment'
plotRowTree(
  object,
  tree_name = "phylo",
  relabel_tree = FALSE,
  order_tree = FALSE,
  remove_levels = FALSE,
  show_label = FALSE,
  show_highlights = FALSE,
  show_highlight_label = FALSE,
  abbr_label = FALSE,
  add_legend = TRUE,
  layout = "circular",
  edge_colour_by = NULL,
  edge_size_by = NULL,
tip_colour_by = NULL,
tip_shape_by = NULL,
tip_size_by = NULL,
node_colour_by = NULL,
node_shape_by = NULL,
node_size_by = NULL,
colour_highlights_by = NULL,
by_exprs_values = "counts",
other_fields = list(),
...
)
```

**Arguments**

- **object**: a `TreeSummarizedExperiment` object.
- **...**: additional arguments for plotting. See `mia-plot-args` for more details i.e. call `help("mia-plot-args")`
- **tree_name**: a single character value specifying a rowTree/colTree from `object`. (By default: `tree_name = "phylo"`)
plotTree

relabel_tree logical scalar. Should the tip labels be relabeled using the output of getTaxonomyLabels(object, with_rank = TRUE)? (default: relabel_tree = FALSE)

order_tree logical scalar. Should the tree be ordered based on alphabetic order of taxonomic levels? (default: order_tree = FALSE)

remove_levels logical scalar. Should taxonomic level information be removed from labels? (default: remove_levels = FALSE)

show_label, show_highlights, show_highlight_label, abbr_label

logical (scalar), integer or character vector. If a logical scalar is given, should tip labels be plotted or if a logical vector is provided, which labels should be shown? If an integer or character vector is provided, it will be converted to a logical vector. The integer values must be in the range of 1 and number of nodes, whereas the values of a character vector must match values of the label column in the node data. In case of a character vector only values corresponding to actual labels will be plotted and if no labels are provided no labels will be shown. (default: FALSE)

add_legend logical scalar. Should legends be plotted? (default: add_legend = TRUE)

layout layout for the plotted tree. See ggtree for details.

datastructure

edge_colour_by Specification of a column metadata field or a feature to colour tree edges by, see the by argument in ?retrieveCellInfo for possible values.

datastructure

edge_size_by Specification of a column metadata field or a feature to size tree edges by, see the by argument in ?retrieveCellInfo for possible values.

datastructure

tip_colour_by Specification of a column metadata field or a feature to colour tree tips by, see the by argument in ?retrieveCellInfo for possible values.

datastructure

tip_shape_by Specification of a column metadata field or a feature to shape tree tips by, see the by argument in ?retrieveCellInfo for possible values.

datastructure

tip_size_by Specification of a column metadata field or a feature to size tree tips by, see the by argument in ?retrieveCellInfo for possible values.

node_colour_by Specification of a column metadata field or a feature to colour tree nodes by. Must be a field from other_fields.

node_shape_by Specification of a column metadata field or a feature to shape tree nodes by. Must be a field from other_fields.

node_size_by Specification of a column metadata field or a feature to size tree nodes by. Must be a field from other_fields.

by_values A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.

other_fields Additional fields to include in the node information without plotting them.

Details

If show_label or show_highlight_label have the same length as the number of nodes, the vector will be used to relabel the nodes.
Value

a `ggtree` plot

See Also

`splitByRanks`

Examples

```r
library(scater)
library(mia)
# preparation of some data
data(GlobalPatterns)
altExps(GlobalPatterns) <- splitByRanks(GlobalPatterns)
altExp(GlobalPatterns,"Genus") <- addPerFeatureQC(altExp(GlobalPatterns,"Genus"))
rowData(altExp(GlobalPatterns,"Genus"))$log_mean <-
  log(rowData(altExp(GlobalPatterns,"Genus"))$mean)
rowData(altExp(GlobalPatterns,"Genus"))$detected <-
  rowData(altExp(GlobalPatterns,"Genus"))$detected / 100
top_genus <- getTopFeatures(altExp(GlobalPatterns,"Genus"),
  method="mean",
  top=100L,
  assay.type="counts")

#
x <- altExp(GlobalPatterns,"Genus")
plotRowTree(x[rownames(x) %in% top_genus,],
  tip_colour_by = "log_mean",
  tip_size_by = "detected")

# plot with tip labels
plotRowTree(x[rownames(x) %in% top_genus,],
  tip_colour_by = "log_mean",
  tip_size_by = "detected",
  show_label = TRUE)

# plot with selected labels
labels <- c("Genus:Providencia", "Genus:Morganella", "0.961.60")
plotRowTree(x[rownames(x) %in% top_genus,],
  tip_colour_by = "log_mean",
  tip_size_by = "detected",
  show_label = labels,
  layout="rectangular")

# plot with labeled edges
plotRowTree(x[rownames(x) %in% top_genus,],
  edge_colour_by = "Phylum",
  tip_colour_by = "log_mean")

# if edges are sized, colours might disappear depending on plotting device
plotRowTree(x[rownames(x) %in% top_genus,],
  edge_colour_by = "Phylum",
  edge_size_by = "detected",
  tip_colour_by = "log_mean")
```
# aggregating data over the taxonomic levels for plotting a taxonomic tree
# please note that the original tree of GlobalPatterns is dropped by
# unsplitByRanks
altExps(GlobalPatterns) <- splitByRanks(GlobalPatterns)
top_phyla <- getTopFeatures(altExp(GlobalPatterns,"Phylum"),
    method="mean",
    top=10L,
    assay.type="counts")
altExps(GlobalPatterns) <- lapply(altExps(GlobalPatterns), addPerFeatureQC)
altExps(GlobalPatterns) <- lapply(altExps(GlobalPatterns),
    function(y){
        rowData(y)$log_mean <- log(rowData(y)$mean)
        rowData(y)$detected <- rowData(y)$detected / 100
        y
    })
x <- unsplitByRanks(GlobalPatterns)
x <- addHierarchyTree(x)
highlights <- c("Phylum:Firmicutes","Phylum:Bacteroidetes",
    "Family:Pseudomonadaceae","Order:Bifidobacteriales")
plotRowTree(x[rowData(x)$Phylum %in% top_phyla,],
    tip_colour_by = "log_mean",
    node_colour_by = "log_mean",
    show_highlights = highlights,
    show_highlight_label = highlights,
    colour_highlights_by = "Phylum")
plotRowTree(x[rowData(x)$Phylum %in% top_phyla,],
    edge_colour_by = "Phylum",
    edge_size_by = "detected",
    tip_colour_by = "log_mean",
    node_colour_by = "log_mean")

---

**treeData**

*Adding information to tree data in TreeSummarizedExperiment*

**Description**

To facilitate the dressing of the tree data stored in a TreeSummarizedExperiment object, rowTreeData and colTreeData can be used.

**Usage**

rowTreeData(x, ...)

colTreeData(x, ...)

rowTreeData(x, tree_name = "phylo") <- value
colTreeData(x, tree_name = "phylo") <- value
combineTreeData(x, other_fields = list())
combineTreeData(x, other_fields = list())

## S4 method for signature 'TreeSummarizedExperiment'
colTreeData(x, tree_name = "phylo")

## S4 method for signature 'TreeSummarizedExperiment'
rowTreeData(x, tree_name = "phylo")

## S4 replacement method for signature 'TreeSummarizedExperiment'
colTreeData(x, tree_name = "phylo") <- value

## S4 replacement method for signature 'TreeSummarizedExperiment'
rowTreeData(x, tree_name = "phylo") <- value

## S4 method for signature 'phylo'
combineTreeData(x, other_fields = list())

## S4 method for signature 'treedata'
combineTreeData(x, other_fields = list())

Arguments

x  a TreeSummarizedExperiment object.
...
additional arguments, currently not used.
tree_name
a single character value specifying a rowTree/colTree from x. (By default:
tree_name = "phylo")
other_fields
a data.frame or coercible to one, with at least one type of id information. See
details.

Details

To match information to nodes, the id information in other_fields are used. These can either be
a column, named ‘node’ or ‘label’ (‘node’ taking precedent), or rownames. If all rownames can
be coerced to integer, they are considered as ‘node’ values, otherwise as ‘label’ values. The id
information must be unique and match available values of rowTreeData(c)

The result of the accessors, rowTreeData and colTreeData, contain at least a ‘node’ and ‘label’
column.

Value

a data.frame for the accessor and the modified TreeSummarizedExperiment object
Examples

```r
data(GlobalPatterns)
td <- rowTreeData(GlobalPatterns)
td
td$test <- rnorm(nrow(td))
rowTreeData(GlobalPatterns) <- td
rowTreeData(GlobalPatterns)
combineTreeData(rowTree(GlobalPatterns), td)
```
Index

* datasets
  mia-datasets, 3
?agglomerateByRank, 21
?retrieveCellInfo, 18, 27
?retrieveFeatureInfo, 21, 22
agglomerateByPrevalence, 22
agglomerateByRank, 21, 22
AsIs, 15
assay, 9, 24
BiocParallelParam, 21
calculateDMN, 16
calculateRDA, 12
col_graph (mia-datasets), 3
ColData, 23
colTreeData (treeData), 29
colTreeData, TreeSummarizedExperiment-method (treeData), 29
colTreeData<-(treeData), 29
collTreeData<-, TreeSummarizedExperiment-method (treeData), 29
combineTreeData (treeData), 29
combineTreeData, phylo-method (treeData), 29
combineTreeData, treedata-method (treeData), 29
facet_wrap, 7
gem_edge Fan, 18
gem_label, 12
gem_label_repel, 12
getPrevalence, 21, 22
ggplot, 7
ggraph, 17
ggtree, 18, 27, 28
metadata, 15
mia, 2

mia-datasets, 3
mia-plot-args, 3
miaViz (miaViz-package), 2
miaViz-package, 2
plotAbundance, 6
plotAbundance, SummarizedExperiment-method (plotAbundance), 6
plotAbundanceDensity, 8
plotAbundanceDensity, SummarizedExperiment-method (plotAbundanceDensity), 8
plotCCA, 11
plotCCA, matrix-method (plotCCA), 11
plotCCA, SingleCellExperiment-method (plotCCA), 11
plotColGraph (plotGraph), 16
plotColGraph, ANY, SummarizedExperiment-method (plotGraph), 16
plotColGraph, SummarizedExperiment, missing-method (plotGraph), 16
plotColTile, 14
plotColTile, SummarizedExperiment-method (plotColTile), 14
plotColTree (plotTree), 25
plotColTree, TreeSummarizedExperiment-method (plotTree), 25
plotDMN, 15
plotDMNFit (plotDMN), 15
plotDMNFit, SummarizedExperiment-method (plotDMN), 15
plotFeaturePrevalence (plotPrevalence), 20
plotFeaturePrevalence, SummarizedExperiment-method (plotPrevalence), 20
plotGraph, 16
plotPrevalence, 20
plotPrevalence, SummarizedExperiment-method (plotPrevalence), 20
plotPrevalentAbundance (plotPrevalence), 20
INDEX

plotPrevalentAbundance, SummarizedExperiment-method
(plotPrevalence), 20
plotRDA (plotCCA), 11
plotRDA, matrix-method (plotCCA), 11
plotRDA, SingleCellExperiment-method
(plotCCA), 11
plotReducedDim, 12, 13
plotRowGraph (plotGraph), 16
plotRowGraph, ANY, SummarizedExperiment-method
(plotGraph), 16
plotRowGraph, SummarizedExperiment, missing-method
(plotGraph), 16
plotRowTile (plotColTile), 14
plotRowTile, SummarizedExperiment-method
(plotColTile), 14
plotRowTree (plotTree), 25
plotRowTree, TreeSummarizedExperiment-method
(plotTree), 25
plotSeries, 23
plotSeries, SummarizedExperiment-method
(plotSeries), 23
plotTaxaPrevalence (plotPrevalence), 20
plotTaxaPrevalence, ANY-method
(plotPrevalence), 20
plotTree, 25
retrieveCellInfo, 15
retrieveFeatureInfo, 15
row_graph (mia-datasets), 3
row_graph_order (mia-datasets), 3
rownames, 24
rowTreeData (treeData), 29
rowTreeData, TreeSummarizedExperiment-method
(treeData), 29
rowTreeData<- (treeData), 29
rowTreeData<-, TreeSummarizedExperiment-method
(treeData), 29
runRDA, 12
splitByRanks, 28
SummarizedExperiment, 6, 9, 15, 17, 18, 21, 23

treeData, 29
TreeSummarizedExperiment, 12, 13, 26, 30