Package ‘MatrixQCvis’

January 19, 2024

Type Package
Title Shiny-based interactive data-quality exploration for omics data
Version 1.10.0
Date 2023-07-19
VignetteBuilder knitr

Description Data quality assessment is an integral part of preparatory data analysis to ensure sound biological information retrieval.

We present here the MatrixQCvis package, which provides shiny-based interactive visualization of data quality metrics at the per-sample and per-feature level. It is broadly applicable to quantitative omics data types that come in matrix-like format (features x samples). It enables the detection of low-quality samples, drifts, outliers and batch effects in data sets. Visualizations include amongst others bar- and violin plots of the (count/intensity) values, mean vs standard deviation plots, MA plots, empirical cumulative distribution function (ECDF) plots, visualizations of the distances between samples, and multiple types of dimension reduction plots. Furthermore, MatrixQCvis allows for differential expression analysis based on the limma (moderated t-tests) and proDA (Wald tests) packages. MatrixQCvis builds upon the popular Bioconductor SummarizedExperiment S4 class and enables thus the facile integration into existing workflows. The package is especially tailored towards metabolomics and proteomics mass spectrometry data, but also allows to assess the data quality of other data types that can be represented in a SummarizedExperiment object.

Depends SummarizedExperiment (>= 1.20.0), plotly (>= 4.9.3), shiny (>= 1.6.0)
Imports ComplexHeatmap (>= 2.7.9), dplyr (>= 1.0.5), ExperimentHub (>= 2.6.0), ggplot2 (>= 3.3.3), grDevices (>= 4.1.0), Hmisc (>= 4.5-0), htmlwidgets (>= 1.5.3), impute (>= 1.65.0), imputeLCMD (>= 2.0), limma (>= 3.47.12), MASS (>= 7.3-58.1), methods (>= 4.1.0), pcaMethods (>= 1.83.0), proDA (>= 1.5.0), rlang (>= 0.4.10), rmarkdown (>= 2.7), Rtsne (>= 0.15), shinydashboard (>= 0.7.1), shinyhelper (>= 0.3.2), shinyjs (>= 2.0.0), stats (>= 4.1.0), tibble (>= 3.1.1), tidyr (>= 1.1.3), umap (>= 0.2.7.0), UpSetR (>= 1.4.0), vsn (>= 3.59.1)
**R topics documented:**

- barplotSamplesMeasuredMissing ........................................... 3
- batchCorrectionAssay ....................................................... 4
- createBoxplot ............................................................... 5
- createDiffFeature ........................................................... 6
- cv ............................................................................. 7
- cvFeaturePlot ............................................................... 8
- dimensionReduction ......................................................... 8
- dimensionReductionPlot ..................................................... 10
- distSample ................................................................. 11
- distShiny ................................................................. 12
- driftPlot ................................................................. 13
- ECDF ................................................................. 14
- explVar ................................................................. 15
- extractComb ............................................................... 16
- featurePlot ............................................................... 17
- histFeature ............................................................... 18
- histFeatureCategory ...................................................... 18
- hist_sample ............................................................. 19
- hist_sample_num ........................................................... 20
- hoeffDPlot ............................................................... 21
- hoeffDValues .............................................................. 22
- imputeAssay ............................................................... 23
- MAplot ................................................................. 24
- MAvalues ................................................................. 25
barplotSamplesMeasuredMissing

Barplot of number of measured/missing features of samples

Description

barplotSamplesMeasuredMissing plots the number of measured/missing features of samples as a barplot. The function will take as input the returned tbl of samplesMeasuredMissing.

Usage

barplotSamplesMeasuredMissing(tbl, measured = TRUE)

Arguments

tbl tbl object
measured logical, should the number of measured or missing values be plotted

Value

gg object from ggplot2

Examples

## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
batchCorrectionAssay

Remove batch effects from (count/intensity) values of a SummarizedExperiment

Description

The function batchCorrectionAssay removes the batch effect of (count/intensity) values of a SummarizedExperiment. It uses either the removeBatchEffect function or no batch effect correction method (pass-through, none).

Usage

batchCorrectionAssay(
  se,
  method = c("none", "removeBatchEffect (limma)",
  batchColumn = colnames(colData(se))
)

Arguments

se SummarizedExperiment
method character, one of "none" or "removeBatchEffect"
batchColumn character, one of colnames(colData(se))

Details

The column batchColumn in colData(se) contains the information on the batch identity. Internal use in shinyQC.

If batchColumn is NULL, batchColumn is internally set to the name of the first column in colData(se) if method = "removeBatchEffect (limma)".

Value

matrix
createBoxplot

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
        dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a),
        type = c(rep("1", 5), rep("2", 5)), batch = rep(c(1, 2), 5))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
        rowData = rD, colData = cD)

batchCorrectionAssay(se, method = "removeBatchEffect (limma)",
                    batchColumn = "batch")
```

createBoxplot

Create a boxplot of (count/intensity) values per sample

Description

The function `create_boxplot` creates a boxplot per sample for the intensity/count values.

Usage

```r
createBoxplot(
        se,
        orderCategory = colnames(colData(se)),
        title = "",
        log = TRUE,
        violin = FALSE
    )
```

Arguments

- `se` SummarizedExperiment containing the (count/intensity) values in the assay slot
- `orderCategory` character, one of `colnames(colData(se))`
- `title` character or numeric of length(1)
- `log` logical, if TRUE (count/intensity) values are displayed as log values
- `violin` logical, if FALSE a boxplot is created, if TRUE a violin plot is created

Details

Internal usage in shinyQC.
createDfFeature

Create data frame of (count/intensity) values for a selected feature along data processing steps

Value

gg object from ggplot2

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
        rowData = rD, colData = cD)
createBoxplot(se, orderCategory = "name", title = "", log = TRUE,
        violin = FALSE)
```

createDfFeature Create data frame of (count/intensity) values for a selected feature along data processing steps

Description

The function createDfFeature takes as input a list of matrices and returns the row Feature of each matrix as a column of a data.frame. The function createDfFeature provides the input for the function featurePlot.

Usage

```r
createDfFeature(l, feature)
```

Arguments

- `l` list containing matrices at different processing steps
- `feature` character, element of rownames of the matrices in l

Details

Internal usage in shinyQC

Value

data.frame
Examples

set.seed(1)
x1 <- matrix(rnorm(100), ncol = 10, nrow = 10,
  dimnames = list(paste("feature", 1:10), paste("sample", 1:10)))
x2 <- x1 + 5
x3 <- x2 + 10

l <- list(x1 = x1, x2 = x2, x3 = x3)
createDfFeature(l, "feature 1")

---

cv  

Calculate coefficient of variation

Description

The function cv calculates the coefficient of variation from columns of a matrix. The coefficients of variation are calculated according to the formula \( \frac{sd(y)}{mean(y)} \times 100 \) with \( y \) the column values, thus, the function returns the coefficient of variation in percentage.

Usage

\( cv(x, name = "raw") \)

Arguments

- **x**: matrix
- **name**: character, the name of the returned list

Details

The function returned a named list (the name is specified by the name argument) containing the coefficient of variation of the columns of \( x \).

Value

list

Examples

\( x <- matrix(1:10, ncol = 2) \)
\( cv(x) \)
cvFeaturePlot  
Plot of feature-wise coefficient of variation values

Description

The function cvFeaturePlot returns a plotly plot of coefficient of variation values. It will create a violin plot and superseeded points of coefficient of variation values per list entry of l.

Usage

   cvFeaturePlot(l, lines = FALSE)

Arguments

l  list containing matrices
lines  logical

Details

lines = TRUE will connect the points belonging to the same feature with a line. If there are less than two features, the violin plot will not be plotted. The violin plots will be ordered according to the order in l

Value

plotly

Examples

   x1 <- matrix(1:100, ncol = 10, nrow = 10,
                dimnames = list(paste("feature", 1:10), paste("sample", 1:10)))
   x2 <- x1 + 5
   x3 <- x2 + 10
   l <- list(x1 = x1, x2 = x2, x3 = x3)
   cvFeaturePlot(l, lines = FALSE)

dimensionReduction  
Dimensionality reduction with dimensionReduction methods PCA, PCoA, NMDS, UMAP and tSNE
**Dimension Reduction**

**Description**

The function `dimensionReduction` creates a data frame with the coordinates of the projected data (first entry of returned output). The function allows for the following projections: Principal Component Analysis (PCA), Principal Coordinates Analysis/Multidimensional Scaling (PCoA), Non-metric Multidimensional scaling (NMDS), t-distributed stochastic neighbor embedding (tSNE), and Uniform Manifold Approximation and Projection (UMAP).

The second list entry will contain the object returned from `prcomp` (PCA), `cmdscale` (PCoA), `isoMDS` (NMDS), `Rtsne` (tSNE), or `umap` (UMAP).

**Usage**

```r
dimensionReduction(
  x,
  type = c("PCA", "PCoA", "NMDS", "tSNE", "UMAP"),
  params = list()
)
```

**Arguments**

- `x` matrix, containing no missing values, samples are in columns and features are in rows
- `type` character, specifying the type/method to use for dimensionality reduction. One of PCA, PCoA, NMDS, tSNE, or UMAP.
- `params` list, arguments/parameters given to the functions `stats::prcomp`, `stats::dist`, `Rtsne::Rtsne`, `umap::umap`

**Details**

The function `dimensionReduction` is a wrapper around the following functions `stats::prcomp` (PCA), `stats::cmdscale` (PCoA), `MASS::isoMDS` (NMDS), `Rtsne::Rtsne` (tSNE), and `umap::umap` (UMAP). For the function `umap::umap` the method is set to naive.

**Value**

list, first entry contains a tbl, second entry contains the object returned from `prcomp` (PCA), `cmdscale` (PCoA), `isoMDS` (NMDS), `Rtsne` (tSNE), or `umap` (UMAP)

**Author(s)**

Thomas Naake

**Examples**

```r
x <- matrix(rnorm(1:10000), ncol = 100)
rownames(x) <- paste("feature", 1:nrow(x))
colnames(x) <- paste("sample", 1:ncol(x))
params <- list(method = "euclidean", ## dist
               initial_dims = 10, max_iter = 100, dims = 3, perplexity = 3, ## tSNE
               min_dist = 0.1, n_neighbors = 15, spread = 1) ## UMAP
```
dimensionReductionPlot

Plot the coordinates from dimensionReduction values

Description

The function `dimensionReductionPlot` creates a dimension reduction plot. The function takes as input the `tbl` object obtained from the `dimensionReduction` function. The `tbl` contains transformed values by one of the dimension reduction methods.

Usage

```r
dimensionReductionPlot(
  tbl, se,
  highlight = c("none", colnames(se@colData)),
  explainedVar = NULL,
  x_coord, y_coord,
  height = 600,
  interactive = TRUE
)
```

Arguments

- `tbl`: `tbl` as obtained by the function `dimensionReduction`
- `se`: `SummarizedExperiment`
- `highlight`: character, one of "none" or `colnames(se@colData)`
- `explainedVar`: `NULL` or named numeric, if numeric `explainedVar` contains the explained variance per principal component (names of `explainedVar` corresponds to the principal components)
- `x_coord`: character, column name of `tbl` that stores x coordinates
- `y_coord`: character, column name of `tbl` that stores y coordinates
- `height`: numeric, specifying the height of the plot (in pixels)
- `interactive`: logical(1), if `TRUE` `dimensionReductionPlot` will return a plotly object, if `FALSE` `dimensionReductionPlot` will return a gg object
Details
The function `dimensionReductionPlot` is a wrapper for a ggplot/ggplotly expression.

Value
plotly or gg

Author(s)
Thomas Naake

Examples
```r
library(SummarizedExperiment)

## create se
a <- matrix(1:100, nrow = 10, ncol = 10, byrow = TRUE,
            dimnames = list(1:10, paste("sample", 1:10)))
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment(assay = a, rowData = rD, colData = cD)
pca <- dimensionReduction(x = assay(se), type = "PCA", params = list())[1]
dimensionReductionPlot(tbl = pca, se = se, highlight = "type",
                        x_coord = "PC1", y_coord = "PC2")
```

---

**distSample**

Create a heatmap using distance information between samples

Description
The function `distSample` creates a heatmap from a distance matrix created by the function `distShiny`. The heatmap is annotated by the column specified by the label column in `colData(se)`.

Usage
```
distSample(d, se, label = "name", title = "raw", ...)
```

Arguments
- `d`: matrix containing distances, obtained from `distShiny`
- `se`: `SummarizedExperiment`
- `label`: character, refers to a column in `colData(se)`
- `title`: character
- `...`: further arguments passed to `ComplexHeatmap::Heatmap`
distShiny

Create distance matrix from numerical matrix

Description

The function distShiny takes as an input a numerical matrix or data.frame and returns the distances between the rows and columns based on the defined method (e.g. euclidean distance).

Usage

distShiny(x, method = "euclidean")

Arguments

  x          matrix or data.frame with samples in columns and features in rows
  method     character, method for distance calculation

Details

  Internal use in shinyQC.
driftPlot

Value
matrix

Examples
x <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
distShiny(x = x)

driftPlot  
Plot the trend line for aggregated values

Description
The function driftPlot aggregates the (count/intensity) values from the assay() slot of a SummarizedExperiment by the median or sum of the (count/intensity) values. driftPlot then visualizes these aggregated values and adds a trend line (using either LOESS or a linear model) from (a subset of) the aggregated values. The subset is specified by the arguments category and level.

Usage
driftPlot(
  se,
  aggregation = c("median", "sum"),
  category = colnames(colData(se)),
  orderCategory = colnames(colData(se)),
  level = c("all", unique(colData(se)[, category])),
  method = c("loess", "lm")
)

Arguments
se SummarizedExperiment
aggregation character, type of aggregation of (count/intensity) values
category character, column of colData(se)
orderCategory character, column of colData(se)
level character, from which samples should the LOESS curve be calculated, either "all" or one of the levels of the selected columns of colData(se) ("category")
method character, either "loess" or "lm"

Details
The x-values are sorted according to the orderCategory argument: The levels of the corresponding column in colData(se) are pasted with the sample names (in the column name) and factorized. Internal usage in shinyQC.
Value

 gg object from ggplot2

Examples

#' ## create se
set.seed(1)
  a <- matrix(rnorm(1000), nrow = 10, ncol = 100,
  dimnames = list(1:10, paste("sample", 1:100)))
a[c(1, 5, 8), 1:5] <- NA
  cD <- data.frame(name = colnames(a), type = c(rep("1", 50), rep("2", 50)))
rD <- data.frame(spectra = rownames(a))
  se <- SummarizedExperiment::SummarizedExperiment(assay = a,
  rowData = rD, colData = cD)

  driftPlot(se, aggregation = "sum", category = "type",
    orderCategory = "type", level = "1", method = "loess")

---

ECDF

Create ECDF plot of a sample against a reference

Description

The function ECDF creates a plot of the empirical cumulative distribution function of a specified sample and an outgroup (reference). The reference is specified by the group argument. The row-wise (feature) mean values of the reference are calculated after excluding the specified sample.

Usage

ECDF(se, sample = colnames(se), group = c("all", colnames(colData(se))))

Arguments

se SummarizedExperiment object
sample character, name of the sample to compare against the group
group character, either "all" or one of colnames(colData(se))

Details

Internal use in shinyQC.

The function ECDF uses the ks.test function from stats to perform a two-sample Kolmogorov-Smirnov test. The Kolmogorov-Smirnov test is run with the alternative "two.sided" (null hypothesis is that the true distribution function of the sample is equal to the hypothesized distribution function of the group).

The exact argument in ks.test is set to NULL, meaning that an exact p-value is computed if the product of the sample sizes is less than 10000 of sample and group. Otherwise, asymptotic distributions are used whose approximations might be inaccurate in low sample sizes.
## explVar

gg object from ggplot2

### Examples

```r
## create se
set.seed(1)
a <- matrix(rnorm(1000), nrow = 100, ncol = 10,
dimnames = list(1:100, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment(assay = a, rowData = rD, colData = cD)

ECDF(se, sample = "sample 1", group = "all")
```

---

**explVar**

Retrieve the explained variance for each principal component (PCA) or axis (PCoA)

### Description

The function `explVar` calculates the proportion of explained variance for each principal component (PC, type = "PCA") and axis (type = "PCoA").

### Usage

```r
explVar(d, type = c("PCA", "PCoA"))
```

### Arguments

- **d**: `prcomp` or list from `cmdscale`
- **type**: character, one of "PCA" or "PCoA"

### Details

`explVar` uses the function `prcomp` from the `stats` package to retrieve the explained standard deviation per PC (type = "PCA") and the function `cmdscale` from the `stats` package to retrieve the explained variation based on eigenvalues per Axis (type = "PCoA").

### Value

numeric vector with the proportion of explained variance for each PC or Axis

### Author(s)

Thomas Naake
Examples

```r
x <- matrix(1:100, nrow = 10, ncol = 10,
            dimnames = list(1:10, paste("sample", 1:10)))
set.seed(1)
x <- x + rnorm(100)

## run for PCA
pca <- dimensionReduction(x = x,
                          params = list(center = TRUE, scale = TRUE), type = "PCA")
explVar(d = pca, type = "PCA")

## run for PCoA
pcoa <- dimensionReduction(x = x,
                          params = list(method = "euclidean"), type = "PCoA")
explVar(d = pcoa, type = "PCoA")
```

---

**extractComb**

Obtain the features that are present in a specified set

### Description

The function `extractComb` extracts the features that match a combination depending if the features was measured or missing. The function will return the sets that match the combination, thus, the function might be useful when answering questions about which features are measured/missing under certain combinations (e.g. sample types or experimental conditions).

### Usage

```r
extractComb(se, combination, measured = TRUE, category = "type")
```

### Arguments

- `se` SummarizedExperiment
- `combination` character, refers to factors in category
- `measured` logical
- `category` character, corresponding to a column name in colData(se)

### Details

The function `extractComb` uses the `make_comb_mat` function from `ComplexHeatmap` package.

Presence is defined by a feature being measured in at least one sample of a set.

Absence is defined by a feature with only missing values (i.e. no measured values) of a set.

### Value

character
Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
  dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a, rowData = rD, colData = cD)
extractComb(se, combination = "2", measured = TRUE, category = "type")
```

---

**featurePlot**

Create a plot of (count/intensity) values over the samples

---

**Description**

The function `featurePlot` creates a plot of (count/intensity) values for different data processing steps (referring to columns in the data.frame) over the different samples (referring to rows in the data.frame).

**Usage**

```r
featurePlot(df)
```

**Arguments**

- `df` data.frame

**Details**

Internal usage in shinyQC.

**Value**

`gg` object from ggplot2

**Examples**

```r
set.seed(1)
x1 <- matrix(rnorm(100), ncol = 10, nrow = 10,
  dimnames = list(paste("feature", 1:10), paste("sample", 1:10)))
x2 <- x1 + 5
x3 <- x2 + 10
l <- list(x1 = x1, x2 = x2, x3 = x3)
df <- createDfFeature(l, "feature 1")
featurePlot(df)
```
The function `histFeature` creates a histogram with the number of measured/missing values per feature.

**Usage**

```
histFeature(x, measured = TRUE, ...)  
```

**Arguments**

- `x` matrix containing intensities. Missing values are encoded as NA.
- `measured` logical, should the measured values (measured = TRUE) or missing values (measured = FALSE) be taken
- `...` additional parameters passed to `geom_histogram`, e.g. `binwidth`.

**Value**

plotly object from `ggplotly`

**Examples**

```
x <- matrix(c(c(1, 1, 1), c(1, NA, 1), c(1, NA, 1),  
c(1, 1, 1), c(NA, 1, 1), c(NA, 1, 1)), byrow = FALSE, nrow = 3)  
colnames(x) <- c("A_1", "A_2", "A_3", "B_1", "B_2", "B_3")  
histFeature(x, binwidth = 1)  
```

The function `histFeatureCategory` creates histogram plots for each sample type in `se`.

**Usage**

```
histFeatureCategory(se, measured = TRUE, category = "type", ...)  
```
Arguments

- `se` SummarizedExperiment, the assay slot contains the intensity values per sample. Missing values are encoded as NA.
- `measured` logical, should the measured values (measured = TRUE) or missing values (measured = FALSE) be taken.
- `category` character, corresponding to a column in `colData(se)`
- `...` additional parameters passed to `geom_histogram`, e.g. `binwidth`.

Value

- plotly object from `ggplotly`

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

histFeatureCategory(se, measured = TRUE, category = "type")
```

**hist_sample**

Plot a histogram of the number of a category

Description

`hist_sample` plots the number of a category (e.g. sample types) as a histogram. It uses the returned `tbl` from `hist_sample_num`.

Usage

`hist_sample(tbl, category = "type")`

Arguments

- `tbl` tbl as returned by `hist_sample_num`
- `category` character, x-axis label of the plot

Value

- gg object from `ggplot2`
Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 4), rep("2", 6)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

tbl <- hist_sample_num(se, category = "type")
hist_sample(tbl)
```

hist_sample_num

Return the number of a category

Description

hist_sample_num returns the number of a category (e.g. sample types) as a tbl. The function will retrieve first the column category in colData(se). The function will return a tbl containing the numerical values of the quantities.

Usage

```r
hist_sample_num(se, category = "type")
```

Arguments

- `se` SummarizedExperiment object
- `category` character, corresponding to a column in colData(se)

Value

- `tbl`

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 4), rep("2", 6)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

tbl <- hist_sample_num(se, category = "type")
hist_sample(tbl)
```
\textbf{hoeffDPlot}

\begin{verbatim}
rowData = rD, colData = cD)
hist_sample_num(se, category = "type")
\end{verbatim}

\textbf{Description}

The function hoeffDPlot creates via ggplot a violin plot per factor, a jitter plot of the data points and (optionally) connects the points via lines. hoeffDPlot uses the plotly package to make the figure interactive.

\textbf{Usage}

\begin{verbatim}
hoeffDPlot(df, lines = TRUE)
\end{verbatim}

\textbf{Arguments}

- \textit{df} \hspace{1cm} data.frame containing one or multiple columns containing the Hoeffding’s D statistics
- \textit{lines} \hspace{1cm} logical, should points belonging to the same sample be connected

\textbf{Details}

The function hoeffDPlot will create the violin plot and jitter plot according to the specified order given by the colnames of \textit{df}. hoeffDPlot will thus internally refactor the colnames of the supplied \texttt{data.frame} according to the order of the colnames.

\textbf{Value}

\texttt{gg} object from \texttt{ggplot2}

\textbf{Examples}

\begin{verbatim}
## create se
set.seed(1)
a <- matrix(rnorm(10000), nrow = 1000, ncol = 10,
dimnames = list(1:1000, paste("sample", 1:10))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
rowData = rD, colData = cD)

tbl <- MAvalues(se, log = FALSE, group = "all")
hd_r <- hoeffDValues(tbl, "raw")
\end{verbatim}
## normalized values

```r
se_n <- se
assay(se_n) <- normalizeAssay(a, "sum")
tbl_n <- MAvalues(se_n, log = FALSE, group = "all")
hd_n <- hoeffDValues(tbl_n, "normalized")
```

df <- data.frame(raw = hd_r, normalized = hd_n)
hoeffDPlot(df, lines = TRUE)
hoeffDPlot(df, lines = FALSE)

---

### hoeffDValues

Create values of Hoeffding’s D statistics from M and A values

**Description**

The function creates and returns Hoeffding’s D statistics values from MA values.

In case `sample_n` is set to a numerical value (e.g. 10000), a random subset containing `sample_n` is taken to calculate Hoeffding’s D values to speed up the calculation. In case there are less features than `sample_n`, all features are taken.

**Usage**

```r
hoeffDValues(tbl, name = "raw", sample_n = NULL)
```

**Arguments**

- `tbl` tibble, as obtained from the function `MAvalues`
- `name` character(1), name of the returned list
- `sample_n` numeric(1), number of features (subset) to be taken for calculation of Hoeffding’s D values

**Details**

The function uses the function `hoeffd` from the `Hmisc` package to calculate the values.

**Value**

named list with Hoeffding’s D values per sample

**Examples**

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
            dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
```
imputeAssay

imputeAssay

## normalized values
se_n <- se
assay(se_n) <- normalizeAssay(a, "sum")
tbl_n <- MAvalues(se_n, group = "all")
hoeffDValues(tbl_n, "normalized")

## transformed values
se_t <- se
assay(se_t) <- transformAssay(a, "log")
tbl_t <- MAvalues(se_t, group = "all")
hoeffDValues(tbl_t, "transformed")

**imputeAssay**

*Impute missing values in a matrix*

**Description**

The function *impute* imputes missing values based on one of the following principles: Bayesian missing value imputation (BPCA), k-nearest neighbor averaging (kNN), Maximum likelihood-based imputation method using the EM algorithm (MLE), replacement by the smallest non-missing value in the data (Min), replacement by the minimal value observed as the q-th quantile (MinDet, default q = 0.01), and replacement by random draws from a Gaussian distribution centred to a minimal value (MinProb).

**Usage**

```r
imputeAssay(a, method = c("BPCA", "kNN", "MLE", "Min", "MinDet", "MinProb"))
```

**Arguments**

- `a`: matrix with samples in columns and features in rows
- `method`: character, one of "BPCA", "kNN", "MLE", "Min", "MinDet", or "MinProb"

**Details**

BPCA wrapper for pcaMethods::pca with methods = "bpca". BPCA is a missing at random (MAR) imputation method.

kNN wrapper for impute::impute.knn with k = 10, rowmax = 0.5, colmax = 0.5, maxp = 1500. kNN is a MAR imputation method.
MLE wrapper for `imputeLCMD::impute.MAR` with `method = "MLE", model.selector = 1/`imputeLCMD::impute.wrapper.MLE`.

MLE is a MAR imputation method.

Min imputes the missing values by the observed minimal value of x. Min is a missing not at random (MNAR) imputation method.

MinDet is a wrapper for `imputeLCMD::impute.MinDet` with `q = 0.01`. MinDet performs the imputation using a deterministic minimal value approach. The missing entries are replaced with a minimal value, estimated from the q-th quantile from each sample. MinDet is a MNAR imputation method.

MinProb is a wrapper for `imputeLCMD::impute.MinProb` with `q = 0.01` and `tune.sigma = 1`. MinProb performs the imputation based on random draws from a Gaussian distribution with the mean set to the minimal value of a sample. MinProb is a MNAR imputation method.

Value

matrix

Examples

```r
a <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA

imputeAssay(a, method = "kNN")
imputeAssay(a, method = "Min")
imputeAssay(a, method = "MinDet")
imputeAssay(a, method = "MinProb")
```

MAplot

Create a MA plot

Description

The function creates a 2D histogram of M and A values.

Usage

```r
MAplot(
tbl,
  group = c("all", colnames(tbl)),
  plot = c("all", unique(tbl[["name"]]))
)
```

Arguments

- `tbl`: tibble containing the M and A values, as obtained from the `MAvalues` function
- `group`: character, one of `colnames(colData(se))` (se used in `MAvalues`) or "all"
- `plot`: character, one of `colData(se)$name` (se used in `MAvalues`) or "all"
MAvalues

Details
MAplot returns a 2D hex histogram instead of a classical scatterplot due to computational reasons and better visualization of overlaying points. The argument plot specifies the sample (refering to colData(se)$name) to be plotted. If plot = "all", MA values for all samples will be plotted (samples will be plotted in facets). If the number of features (tbl$Features) is below 1000, points will be plotted (via geom_points), otherwise hexagons will be plotted (via geom_hex).

Value
gg object from ggplot2

Examples
```r
## create se
set.seed(1)
a <- matrix(rnorm(10000), nrow = 1000, ncol = 10,
    dimnames = list(1:1000, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

tbl <- MAvalues(se, log = FALSE, group = "all")
MAplot(tbl, group = "all", plot = "all")
```

Description
The function MAvalues will create MA values as input for the function MAplot and hoeffDValues. M and A are specified relative to specified samples which is determined by the group argument. In case of group == "all", all samples (expect the specified one) are taken for the reference calculation. In case of group != "all" will use the samples belonging to the same group given in colnames(colData(se)) expect the specified one.

Usage
MAvalues(se, log2 = TRUE, group = c("all", colnames(colData(se))))

Arguments
- **se**: SummarizedExperiment
- **log2**: logical, specifies if values are log2-transformed prior to calculating M and A values. If the values are already transformed, log2 should be set to FALSE. If log2 = TRUE and if there are values in assay(se) that are 0, the log2 values are calculated by log2(assay(se) + 1)
### measuredCategory

**Obtain the number of measured intensities per sample type**

The function `measuredCategory` creates a `tbl` with the number of measured values per feature. 0 means that there were only missing values (NA) for the feature and sample type. `measuredCategory` will return a `tbl` where columns are the unique sample types and rows are the features as in `assay(se)`.

#### Usage

```r
measuredCategory(se, measured = TRUE, category = "type")
```

#### Arguments

- `se` (SummarizedExperiment)
- `measured` (logical): should the measured values (`measured = TRUE`) or missing values (`measured = FALSE`) be taken
- `category` (character): corresponds to a column name in `colData(se)`

#### Details

`measuredCategory` is a helper function.

#### Value

matrix with number of measured/missing features per category type
Examples

```r
## create se
set.seed(1)
a <- matrix(rnorm(100), nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a),
    type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)
measuredCategory(se, measured = TRUE, category = "type")
```

---

### mosaic

**Mosaic plot for two factors in colData(se)**

Description

The function `mosaic` creates a mosaic plot of two factors from an `SummarizedExperiment` object. The columns `f1` and `f2` are taken from `colData(se)`.

Usage

```r
mosaic(se, f1, f2)
```

Arguments

- `se` SummarizedExperiment object
- `f1` character, `f1` is one of the column names in `colData(se)`
- `f2` character, `f2` is one of the column names in `colData(se)`

Details


Value

`gg` object from `ggplot2`

Examples

```r
## create se
set.seed(1)
a <- matrix(rnorm(100), nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a),
    type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)
measuredCategory(se, measured = TRUE, category = "type")
```
normalizeAssay

Normalize a data sets (reduce technical sample effects)

Description

The function normalizeAssay performs normalization by sum of the (count/intensity) values per sample or quantile division per sample or by quantile normalization (adjusting the distributions that they become identical in statistical distributions). The divisor for quantile division (e.g., the 75 argument. Quantile normalization is performed by using the normalizeQuantiles function from limma.

Usage

normalizeAssay(
  a,
  method = c("none", "sum", "quantile division", "quantile"),
  probs = 0.75
)

Arguments

a          matrix with samples in columns and features in rows
method     character, one of "none", "sum", "quantile division", "quantile"
probs      numeric, ranging between [0, 1). probs is used as the divisor for quantile division in method = "quantile division"

Details

Internal usage in shinyQC. If method is set to "none", the object x is returned as is (pass-through). If probs is NULL, probs is internally set to "name" if method = "quantile division".

Value

matrix
Examples

\[
a \leftarrow \text{matrix}(1:100, \text{nrow} = 10, \text{ncol} = 10, \\
\quad \text{dimnames} = \text{list}(1:10, \text{paste("sample", 1:10)}))
\]
\[
\text{normalizeAssay}(a, "\text{sum}")
\]

Description

The function \text{permuteExplVar} determines the explained variance of the permuted expression matrix (x). It is used to determine the optimal number of PCs for tSNE.

Usage

\[
\text{permuteExplVar}(x, n = 10, \text{center} = \text{TRUE}, \text{scale} = \text{TRUE}, \text{sample_n} = \text{NULL})
\]

Arguments

- **x**: matrix or \text{data.frame}, samples in columns and features in rows
- **n**: numeric, number of permutation rounds
- **center**: logical, passed to the function \text{explVar}
- **scale**: logical, passed to the function \text{explVar}
- **sample_n**: numeric(1), number of features (subset) to be taken for calculation of permuted explained variance, the top \text{sample_n} varying values based on their standard deviation will be taken

Details

For the input of tSNE, typically, we want to reduce the initial number of dimensions linearly with PCA (used as \text{initial_dims} arguments in the \text{Rtsne} function). The reduced data set is used for feeding into tSNE. By plotting the percentage of variance explained by the Prinical Components (PCs) we can estimate how many PCs we keep as input to tSNE. However, if we select too many PCs, noise will be included as input to tSNE; if we select too few PCs we might loose the important data structures. To get a better understanding how many PCs to include, randomization will be employed and the observed variance will be compared to the permuted variance.

Value

matrix with explained variance

Author(s)

Thomas Naake
Examples

```r
x <- matrix(1:100, nrow = 10, ncol = 10,
  dimnames = list(1:10, paste("sample", 1:10)))
permuteExplVar(x = x, n = 10, center = TRUE, scale = TRUE, sample_n = NULL)
```

---

**Description**

The function `plotCV` displays the coefficient of variation values of set of values supplied in a `data.frame` object. The function will create a plot using the `ggplot2` package and will print the values in the different columns in different colors.

**Usage**

```r
plotCV(df)
```

**Arguments**

- `df` data.frame containing one or multiple columns containing the coefficients of variation

**Details**

Internal usage in shinyQC.

**Value**

`gg` object from `ggplot2`

**Examples**

```r
x1 <- matrix(1:10, ncol = 2)
x2 <- matrix(11:20, ncol = 2)
x3 <- matrix(21:30, ncol = 2)
x4 <- matrix(31:40, ncol = 2)

## calculate cv values
cv1 <- cv(x1, "x1")
cv2 <- cv(x2, "x2")
cv3 <- cv(x3, "x3")
cv4 <- cv(x4, "x4")

df <- data.frame(cv1, cv2, cv3, cv4)
plotCV(df)
```
Description

The function `plotPCALoadings` creates a loadings plot of the features.

Usage

```r
plotPCALoadings(tbl, x_coord, y_coord)
```

Arguments

- `tbl`: `tbl` as obtained by the function `dimensionReduction`
- `x_coord`: character, column name of `tbl` that stores x coordinates
- `y_coord`: character, column name of `tbl` that stores y coordinates

Details

The function takes as input the output of the function `tblPCALoadings`. It uses the `ggplotly` function from `plotly` to create an interactive `plotly` plot.

Value

`plotly`

Author(s)

Thomas Naake

Examples

```r
x <- matrix(rnorm(1:10000), ncol = 100)
rownames(x) <- paste("feature", 1:nrow(x))
colnames(x) <- paste("sample", 1:ncol(x))
params <- list(method = "euclidean", ## dist
               initial_dims = 10, max_iter = 100, dims = 3, perplexity = 3, ## tSNE
               min_dist = 0.1, n_neighbors = 15, spread = 1) ## UMAP
tbl <- tblPCALoadings(x, params)
plotPCALoadings(tbl, x_coord = "PC1", y_coord = "PC2")
```
plotPCAVar

Plot of explained variance against the principal components

Description

The function `plotPCAVar` plots the explained variance (in y-axis against the principal components for the measured and permuted values.

Usage

```r
plotPCAVar(var_x, var_perm = NULL)
```

Arguments

- `var_x` numeric (named numeric vector)
- `var_perm` matrix with the explained variance obtained by permutation (function `permuteExplVar`)

Details

The argument `var_perm` is optional and visualization of permuted values can be omitted by setting `var_perm = NULL`.

Value

`gg` object from `ggplot`

Author(s)

Thomas Naake

Examples

```r
x <- matrix(1:100, ncol = 10)
pca <- dimensionReduction(x = x, params = list(center = TRUE, scale = TRUE),
                         type = "PCA")
var_x <- explVar(d = pca, type = "PCA")
var_perm <- permuteExplVar(x = x, n = 100, center = TRUE, scale = TRUE)
plotPCAVar(var_x = var_x, var_perm = var_perm)
```
plotPCAVarPvalue

Plot p-values for the significance of principal components

Description

The function plotPCAVarPvalue plots the p-values of significances of principal components. Using the visual output, the optimal number of principal components can be selected.

Usage

plotPCAVarPvalue(var_x, var_perm)

Arguments

var_x numeric, measured variances
var_perm matrix, variances obtained by permutation

Details

Internal usage in shinyQC.

Value

gg object from ggplot

Author(s)

Thomas Naake

Examples

x <- matrix(1:100, ncol = 10)
pca <- dimensionReduction(x = x, params = list(center = TRUE, scale = TRUE), type = "PCA")[[2]]
var_x <- explVar(d = pca, type = "PCA")
var_perm <- permuteExplVar(x = x, n = 100, center = TRUE, scale = TRUE)
plotPCAVarPvalue(var_x = var_x, var_perm = var_perm)
samplesMeasuredMissing

Create tibble containing number of measured/missing features of samples

Description

samplesMeasuredMissing returns a tbl with the number of measured/missing features of samples. The function will take as input a SummarizedExperiment object and will access its assay() slot.

Usage

samplesMeasuredMissing(se)

Arguments

se SummarizedExperiment object

Value

tbl with number of measured/missing features per sample

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
  dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
sample <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
featData <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
  rowData = featData, colData = sample)

## create the data.frame with information on number of measured/missing
## values
samplesMeasuredMissing(se)
```
shinyQC

Shiny application for initial QC exploration of -omics data sets

Description
The shiny application allows to explore -omics data sets especially with a focus on quality control. shinyQC gives information on the type of samples included (if this was previously specified within the SummarizedExperiment object). It gives information on the number of missing and measured values across features and across sets (e.g. quality control samples, control, and treatment groups, only displayed for SummarizedExperiment objects that contain missing values).

shinyQC includes functionality to display (count/intensity) values across samples (to detect drifts in intensity values during the measurement), to display mean-sd plots, MA plots, ECDF plots, and distance plots between samples. shinyQC includes functionality to perform dimensionality reduction (currently limited to PCA, PCoA, NMDS, tSNE, and UMAP). Additionally, it includes functionality to perform differential expression analysis (currently limited to moderated t-tests and the Wald test).

Usage

shinyQC(se, app_server = FALSE)

Arguments

se SummarizedExperiment object (can be omitted)
app_server logical (set to TRUE if run under a server environment)

Details
rownames(se) should be set to the corresponding name of features, while colnames(se) should be set to the sample IDs. rownames(se) and colnames(se) are not allowed to be NULL. colnames(se), colnames(assay(se)) and rownames(colData(se)) all have to be identical.

shinyQC allows to subset the supplied SummarizedExperiment object.

On exit of the shiny application, the (subsetted) SummarizedExperiment object is returned with information on the processing steps (normalization, transformation, batch correction and imputation). The object will only returned if app_server = FALSE and if the function call is assigned to an object, e.g. tmp <- shinyQC(se).

If the se argument is omitted the app will load an interface that allows for data upload.

Value

shiny application, SummarizedExperiment upon exiting the shiny application

Author(s)

Thomas Naake
Examples

```
library(dplyr)
library(SummarizedExperiment)

## create se
set.seed(1)
a <- matrix(rnorm(100, mean = 10, sd = 2), nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment(assay = a, rowData = rD, colData = cD)

shinyQC(se)
```

### sumDistSample

Plot the sum of distances to other samples

Description

The function `sumDistSample` creates a plot showing the sum of distance of a sample to other samples.

Usage

```
sumDistSample(d, title = "raw")
```

Arguments

- **d**  
  matrix containing distances, obtained from `distShiny`
- **title**  
  character specifying the title to be added to the plot

Value

- `gg` object from `ggplot2`

Examples

```
a <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
dist <- distShiny(a)
sumDistSample(dist, title = "raw")
```
tblPCALoadings

Return tibble with PCA loadings for features

Description

The function tblPCALoadings returns a tibble with loadings values for the features (row entries) in x.

Usage

tblPCALoadings(x, params)

Arguments

x matrix, containing no missing values
params list, arguments/parameters given to the function stats::prcomp

Details

The function tblPCALoadings accesses the list entry rotation of the prcomp object.

Value

tbl

Author(s)

Thomas Naake

Examples

set.seed(1)
x <- matrix(rnorm(1:10000), ncol = 100)ownames(x) <- paste("feature", 1:nrow(x))
colnames(x) <- paste("sample", 1:ncol(x))
params <- list(method = "euclidean", ## dist
               initial_dims = 10, max_iter = 100, dims = 3, perplexity = 3, ## tSNE
               min_dist = 0.1, n_neighbors = 15, spread = 1) ## UMAP
tblPCALoadings(x, params)
transformAssay

Transform the (count/intensity) values of a data.frame, tbl or matrix

Description

The function transformAssay transforms the (count/intensity) values of a matrix. It uses either log, log2, variance stabilizing normalisation (vsn) or no transformation method (pass-through, none). The object x has the samples in the columns and the features in the rows.

Usage

transformAssay(x, method = c("none", "log", "log2", "vsn"), .offset = 1)

Arguments

a matrix with samples in columns and features in rows
method character, one of "none", "log", "log2" or "vsn"
.offset numeric(1), offset to add when method set to "log" or "log2" and a contains values of 0, default to 1

Details

Internal use in shinyQC.

Value

matrix

Examples

a <- matrix(1:1000, nrow = 100, ncol = 10,
            dimnames = list(1:100, paste("sample", 1:10)))
transformAssay(a, "none")
transformAssay(a, "log")
transformAssay(a, "log2")
transformAssay(a, "vsn")
Description

The function `upsetCategory` displays the frequency of measured values per feature with respect to class/sample type to assess difference in occurrences. Internally, the measured values per sample are obtained via the `measuredCategory` function: this function will access the number of measured/missing values per category and feature. From this, a binary tbl will be created specifying if the feature is present/missing, which will be given to the `upset` function from the UpSetR package.

Usage

```
upsetCategory(se, category = colnames(colData(se)), measured = TRUE)
```

Arguments

- `se` SummarizedExperiment, containing the intensity values in `assay(se)`, missing values are encoded by `NA`
- `category` character, corresponding to a column in `colData(se)`
- `measured` logical, should the measured values (`measured = TRUE`) or missing values (`measured = FALSE`) be taken

Details

Presence is defined by a feature being measured in at least one sample of a set.
Absence is defined by a feature with only missing values (i.e. no measured values) of a set.

Value

`upset plot`

Examples

```
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
cD <- data.frame(name = colnames(a), type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
rowData = rD, colData = cD)

upsetCategory(se, category = "type")
```
Description

The function `ComplexHeatmap` creates a volcano plot. On the y-axis the -log10(p-values) are displayed, while on the x-axis the fold changes/differences are displayed. The output of the function differs depending on the type parameter. For type == "ttest", the fold changes are plotted; for type == "proDA", the differences are plotted.

Usage

volcanoPlot(df, type = c("ttest", "proDA"))

Arguments

df  data.frame as received from `topTable` (ttest) or `test_diff` (proDA)
type  character

Details

Internal use in `shinyQC`.

Value

plotly

Examples

```r
## create se
a <- matrix(1:100, nrow = 10, ncol = 10,
  dimnames = list(1:10, paste("sample", 1:10)))
a[c(1, 5, 8), 1:5] <- NA
set.seed(1)
a <- a + rnorm(100)
a_i <- imputeAssay(a, method = "MinDet")
cD <- data.frame(sample = colnames(a),
  type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
  rowData = rD, colData = cD)
se_i <- SummarizedExperiment::SummarizedExperiment(assay = a_i,
  rowData = rD, colData = cD)

## create model and contrast matrix
modelMatrix_expr <- stats::formula("~ 0 + type")
contrast_expr <- "type1-type2"
modelMatrix <- model.matrix(modelMatrix_expr, data = colData(se))
contrastMatrix <- limma::makeContrasts(contrasts = contrast_expr,
```
## ttest

```r
fit <- limma::lmFit(a_i, design = modelMatrix) 
fit <- limma::contrasts.fit(fit, contrastMatrix) 
fit <- limma::eBayes(fit, trend = TRUE) 
df_ttest <- limma::topTable(fit, n = Inf, adjust = "fdr", p = 0.05) 
df_ttest <- cbind(name = rownames(df_ttest), df_ttest)
```

## plot

```r
volcanoPlot(df_ttest, type = "ttest")
```

## proDA

```r
fit <- proDA::proDA(a, design = modelMatrix) 
df_proDA <- proDA::test_diff(fit = fit, contrast = contrast_expr, 
                           sort_by = "adj_pval")
```

## plot

```r
volcanoPlot(df_proDA, type = "proDA")
```
Index

barplotSamplesMeasuredMissing, 3
batchCorrectionAssay, 4
createBoxplot, 5
createDfFeature, 6
cv, 7
cvFeaturePlot, 8
dimensionReduction, 8
dimensionReductionPlot, 10
distSample, 11
distShiny, 12
driftPlot, 13
ECDF, 14
explVar, 15
extractComb, 16
featurePlot, 17
hist_sample, 19
hist_sample_num, 20
histFeature, 18
histFeatureCategory, 18
hoeffDPlot, 21
hoeffDValues, 22
imputeAssay, 23
MAplot, 24
MAvalues, 25
measuredCategory, 26
mosaic, 27
normalizeAssay, 28
permuteExp1Var, 29
plotCV, 30
plotPCALoadings, 31
plotPCAVar, 32
plotPCAVarPvalue, 33
samplesMeasuredMissing, 34
shinyQC, 35
sumDistSample, 36
tblPCALoadings, 37
transformAssay, 38
upsetCategory, 39
volcanoPlot, 40