Package 'MatrixQCvis'

April 12, 2022

Type Package

Title Shiny-based interactive data-quality exploration for omics data

Version 1.2.4

Date 2022-04-09

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), 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), methods (>= 4.1.0), openxlsx (>= 4.2.3), pcaMethods (>= 1.83.0), proDA (>= 1.5.0), rlang (>= 0.4.10), rmarkdown (>= 2.7), Rtsne (>= 0.15), S4Vectors (>= 0.29.15), 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), vegan (>= 2.5-7), vsn (>= 3.59.1)

Suggests BiocGenerics (>= 0.37.4), BiocStyle (>= 2.19.2), hexbin (>= 1.28.2), knitr (>= 1.33), testthat (>= 3.0.2)

biocViews Visualization, GUI, DimensionReduction, Metabolomics, Proteomics

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.1

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

git_branch RELEASE_3_14

git_last_commit 9cccc5b

git_last_commit_date 2022-04-09

Date/Publication 2022-04-12

Author Thomas Naake [aut, cre], Wolfgang Huber [aut]

Maintainer Thomas Naake <thomasnaake@googlemail.com>

R topics documented:

`barplot_samples_memi` *Barplot of number of measured/missing features of samples*

Description

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

Usage

```
barplot_samples_memi(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)))
rD <- data.frame(spectra = rownames(a))
```

```
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

## create the data.frame with information on number of measured/missing
## values
tbl <- samples_memi(se)

## plot number of measured values
barplot_samples_memi(tbl, measured = TRUE)

## plot number of missing values
barplot_samples_memi(tbl, measured = FALSE)
```

`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'.

Value

'matrix'

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)), 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")
```

`biocrates` *Convert Biocrates xlsx output to 'SummarizedExperiment' object*

Description

The function 'biocrates' will create a 'SummarizedExperiment' from a Biocrates xlsx file. The function 'biocrates' takes as input the path to a .xlsx file (Biocrates output) and additional parameters given to the 'read.xlsx' function from the 'openxlsx' package (e.g. specifying the sheet name or index by 'sheet').

Usage

```
biocrates(file, sheet, ...)
```

Arguments

`file`	'character'
`sheet`	'character' or 'numeric', the name or index of the sheet to read data from
`...`	additional parameters given to 'read.xslx'

Details

The column "Sample Identification" has to contain unique identifiers (no duplications).

Value

'SummarizedExperiment' object

Examples

```
file <- "path/to/biocrates/object"
biocrates(file = file, sheet = 1)
```

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

```
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")
```

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

Description

The function ‘create_boxplot‘ creates

Usage

```
create_boxplot(
  se,
  orderCategory = colnames(colData(se)),
  title = "",
  log2 = 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)‘
log2	‘logical‘, if ‘TRUE‘ (count/intensity) values are displayed as log2 values
violin	‘logical‘, if ‘FALSE‘ a boxplot is created, if ‘TRUE‘ a violin plot is created

Details

Internal usage in ‘shinyQC‘.

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)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)
```

```
create_boxplot(se, orderCategory = "name", title = "", log2 = TRUE,
    violin = FALSE)
```

---

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 ‘sd(y) / mean(y) * 100‘ with ‘y‘ the column values.

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 superseded 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)
```

`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'

Details

Internal use in 'shinyQC'

Value

'plotly'

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)
a_i <- imputeAssay(a, method = "MinDet")
cD <- data.frame(name = colnames(a_i),
    type = c(rep("1", 5), rep("2", 5)))
rD <- data.frame(spectra = rownames(a_i))
se <- SummarizedExperiment::SummarizedExperiment(assay = a_i, rowData = rD,
    colData = cD)

dist <- distShiny(a_i)
distSample(dist, se, label = "type", title = "imputed",
    show_row_names = TRUE)
```

---

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'.

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‘.

Value

‘gg‘ object from ‘ggplot2‘

Examples

```
## 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

```
explVar(x, params, type = c("PCA", "PCoA"))
```

Arguments

x	‘matrix‘, containing no missing values (‘NA‘), samples in columns and features in rows

params	‘list‘, containing the parameters for PCA and PCoA. For ‘type = "PCA"‘ these are ‘center‘ of type ‘logical‘ (indicating whether the variables should be shifted to be zero centered) and ‘scale‘ of type ‘logical‘(indicating whether the variables should be scaled that they have a standard variation of 1). For ‘type = "PCoA"‘, this is ‘method‘ of type ‘character‘ (indicating the method for distance calculation).
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

```
x <- matrix(1:100, nrow = 10, ncol = 10,
    dimnames = list(1:10, paste("sample", 1:10)))
set.seed(1)
x <- x + rnorm(100)
explVar(x = x, params = list(center = TRUE, scale = TRUE), type = "PCA")
explVar(x = x, params = list(method = "euclidean"), 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

```
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.

Value

'character'

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)

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

```
featurePlot(df)
```

Arguments

df	'data.frame'

Details

Internal usage in ‘shinyQC‘.

Value

‘gg‘ object from ‘ggplot2‘

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)
df <- createDfFeature(l, "feature 1")
featurePlot(df)
```

`hist_feature` *Histogram for measured value per feature*

Description

The function ‘hist_compound‘ creates a histogram with the number of measured/missing values per feature.

Usage

```
hist_feature(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")
hist_feature(x, binwidth = 1)
```

`hist_feature_category` *Histogram of features per sample type*

Description

The function 'hist_feature_category' creates histogram plots for each sample type in 'se'.

Usage

```
hist_feature_category(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

```
## 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)

hist_feature_category(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 use 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

```
## 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

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

Arguments

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

Value

'tbl'

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", 4), rep("2", 6)))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

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

---

hoeffDPlot *Create a plot from a list of Hoeffding's D values*

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.

Usage

```
hoeffDPlot(df, lines = TRUE)
```

Arguments

df	'data.frame' containing one or multiple columns containing the Hoeffding's D statistics
lines	'logical', should points belonging to the same sample be connected

Details

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

Value

‘gg‘ object from ‘ggplot2‘

Examples

```
## 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, log2 = FALSE, group = "all")
hd_r <- hoeffDValues(tbl, "raw")

## normalized values
se_n <- se
assay(se_n) <- normalizeAssay(a, "sum")
tbl_n <- MAvalues(se_n, log2 = 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.

Usage

```
hoeffDValues(tbl, name = "raw")
```

Arguments

tbl	‘tibble‘, as obtained from the function ‘MAvalues‘
name	‘character‘, name of the returned list

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

```
## 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)

tbl <- MAvalues(se)
hoeffDValues(tbl, "raw")

## 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, "log2")
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‘), Malimum 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

```
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 Gaussion distribution with the mean set to the minimal value of a sample. ‘MinProb‘ is a MNAR imputation method.

Value

‘matrix‘

Examples

```
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

```
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"'

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

```
## 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, log2 = FALSE, group = "all")
MAplot(tbl, group = "all", plot = "all")
```

MAvalues *Create values (M and A) for MA plot*

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 re ‘log2‘ transformed prior to calculating M and A values. If the values are already transformed, ‘log2‘ should be set to ‘FALSE‘.
group	‘character‘, either ‘"all"‘ or one of ‘colnames(colData(se))‘

Value

‘tbl‘ with columns ‘Feature‘, ‘name‘ (sample name), ‘A‘, ‘M‘ and additional columns of ‘colData(se)‘

Examples

```
## 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(assay = a, rowData = rD, colData = cD)

MAvalues(se, log2 = FALSE, group = "all")
```

maxQuant *Convert MaxQuant xlsx output to 'SummarizedExperiment' object*

Description

The function 'maxQuant' will create a 'SummarizedExperiment' from a MaxQuant xlsx file. The function 'maxQuant' takes as input the path to a .xlsx file (MaxQuant output) and additional parameters given to the 'read.xlsx' function from the 'openxlsx' package (e.g. specifying the sheet name or index by 'sheet').

Usage

```
maxQuant(file, type = c("iBAQ", "LFQ"), sheet, ...)
```

Arguments

file	'character'
type	'character', either 'iBAQ' or 'LFQ'
sheet	'character' or 'numeric', the name or index of the sheet to read data from
...	additional parameters given to 'read.xslx'

Details

The argument 'type' will specify if the 'iBAQ' or 'LFQ' values are taken.

Value

'SummarizedExperiment' object

Examples

```
file <- "path/to/maxQuant/object"
maxQuant(file = file, type = "iBAQ", sheet = 1)
```

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

Description

The function 'measured_category' 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. 'measured_category' will return a 'tbl' where columns are the unique sample types and rows are the features as in 'assay(se)'.

Usage

```
measured_category(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

'measured_category' is a helper function.

Value

'tbl' with number of measured/mising features per 'category' type

Examples

```
## 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)

measured_category(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 'SummararizedExperiment' object. The columns 'f1' and 'f2' are taken from 'colData(se)'.

Usage

```
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

Code partly taken from https://stackoverflow.com/questions/21588096/pass-string-to-facet-grid-ggplot2

Value

'gg' object from 'ggplot2'

Examples

```
## 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)),
    cell_type = c("A", "B"))
rD <- data.frame(spectra = rownames(a))
se <- SummarizedExperiment::SummarizedExperiment(assay = a,
    rowData = rD, colData = cD)

mosaic(se, "cell_type", "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 Quantile normalization is performed by using the 'normalizeQuantiles' function from 'limma'.

Usage

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

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).

Value

‘matrix‘

Examples

```
a <- matrix(1:100, nrow = 10, ncol = 10,
        dimnames = list(1:10, paste("sample", 1:10)))
normalizeAssay(a, "sum")
```

---

ordination *Dimensionality reduction with ordination methods PCA, PCoA, NMDS, UMAP and tSNE*

Description

The function ‘ordination‘ creates a ‘data.frame‘ with the coordinates of the projected data. 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 Approimation and Projection (UMAP).

Usage

```
ordination(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 ‘ordination‘ is a wrapper around the following functions ‘stats::prcomp‘ (PCA), ‘stats::cmdscale‘ (PCoA), ‘vegan::metaMDS‘ (NMDS), ‘Rtsne::Rtsne‘ (tSNE), and ‘umap::umap‘ (UMAP). For the function ‘umap::umap‘ the method is set to ‘naive‘.

Value

‘tbl‘

Author(s)

Thomas Naake

Examples

```
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
ordination(x, type = "PCA", params = params)
ordination(x, type = "PCoA", params = params)
ordination(x, type = "NMDS", params = params)
ordination(x, type = "tSNE", params = params)
ordination(x, type = "UMAP", params = params)
```

---

ordinationPlot *Plot the coordinates from ‘ordination‘ values*

Description

The function ‘ordinationPlot‘ creates a dimension reduction plot. The function takes as input the ‘tbl‘ object obtained from the ‘ordination‘ function. The ‘tbl‘ contains transformed values by one of the ordination methods.

Usage

```
ordinationPlot(
  tbl,
  se,
  highlight = c("none", colnames(colData(se))),
  explainedVar = NULL,
  x_coord,
  y_coord,
  height = 600
)
```

Arguments

tbl	‘tbl‘ as obtained by the function ‘ordination‘
se	‘SummarizedExperiment‘
highlight	‘character‘, one of ‘"none"‘ or ‘colnames(colData(se))‘
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)

Details

The function ‘ordinationPlot‘ is a wrapper for a ‘ggplot‘/‘ggplotly‘ expression.

Value

‘plotly‘

Author(s)

Thomas Naake

Examples

```
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 <- ordination(x = assay(se), type = "PCA", params = list())

ordinationPlot(tbl = pca, se = se, highlight = "type",
    x_coord = "PC1", y_coord = "PC2")
```

permuteExplVar *Permute the expression values and retrieve the explained variance*

Description

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

Usage

```
permuteExplVar(x, n = 10, center = TRUE, scale = TRUE)
```

Arguments

x	‘matrix‘ or ‘data.frame‘, samples in columns and features in rows
n	‘numeric‘, number of permutation rounds
center	‘logical‘, passed to the function ‘explVar‘
scale	‘logical‘, passed to the function ‘explVar‘

Details

For the input of tSNE, typically, we want to reduce the initial number of dimensions linearly with PCA (used as the ‘initial_dims‘ arguments in the ‘Rtsne‘ funtion). The reduced data set is used for feeding into tSNE. By plotting the percentage of variance explained by the Princical Components (PCs) we can estimate how many PCs we keep as input into 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

```
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)
```

plotCV *Plot CV values*

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

```
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

```
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)
```

plotPCALoadings *Plot for PCA loadings of features*

Description

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

Usage

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

Arguments

tbl	‘tbl‘ as obtained by the function ‘ordination‘
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 ‘tblPlotPCALoadings‘. It uses the ‘ggplotly‘ function from ‘plotly‘ to create an interactive ‘plotly‘ plot.

Value

‘plotly‘

Author(s)

Thomas Naake

Examples

```
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 against the principal components for the measured and permuted values.

Usage

```
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

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

samples_memi *Create tibble containing number of measured/missing features of samples*

Description

'samples_memi' 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

```
samples_memi(se)
```

Arguments

se	'SummarizedExperiment' object

Value

'tbl' with number of measured/missing features per sample

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)
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
samples_memi(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

The ‘se‘ object should not contain a column ‘"rowname"‘ in the ‘colData‘ slot.

‘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.

‘shinyQC‘ allows to subset the supplied ‘SummarizedExperiment‘ object.

On exit of the shiny application (only via the button ’Stop and export data set’), 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‘.

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 (the object will be returned only when exiting the application via the button ’Stop and export data set’)

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‘ acccesses 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)
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
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 '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(a, method = c("none", "log2", "vsn"))
```

Arguments

`a`	'matrix' with samples in columns and features in rows
`method`	'character', one of '"none"', '"log2"' or '"vsn"'

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, "log2")
transformAssay(a, "vsn")
```

`upset_category` *UpSet plot to display measures values across sample types*

Description

The function 'upset_category' 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 'measured_category' 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

```
upset_category(se, category = colnames(colData(se))[1], ...)
```

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)'
...	additional parameters passed to 'measured_category'

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)

upset_category(se, category = "type")
```

---

volcanoPlot *Volcano plot of fold changes/differences against p-values*

Description

The function 'volcanoPlot' 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 'volcanoPlot' 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

```
## 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,
                                levels = modelMatrix)

## ttest
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
volcanoPlot(df_ttest, type = "ttest")

## proDA

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

## plot
volcanoPlot(df_proDA, type = "proDA")
```

Index