Package ‘omicsViewer’

October 11, 2022

Title Interactive and explorative visualization of SummarizedExperssionSet or ExpressionSet using omicsViewer

Version 1.0.1

Description omicsViewer visualizes ExpressionSet (or SummarizedExperiment) in an interactive way. The omicsViewer has a separate back- and front-end. In the backend, users need to prepare an ExpressionSet that contains all the necessary information for the downstream data interpretation. Some extra requirements on the headers of phenotype data or feature data are imposed so that the provided information can be clearly recognized by the front-end, at the same time, keep a minimum modification on the existing ExpressionSet object. The pure dependency on R/Bioconductor guarantees maximum flexibility in the statistical analysis in the back-end. Once the ExpressionSet is prepared, it can be visualized using the front-end, implemented by shiny and plotly. Both features and samples could be selected from (data) tables or graphs (scatter plot/heatmap). Different types of analyses, such as enrichment analysis (using Bioconductor package fgsea or fisher's exact test) and STRING network analysis, will be performed on the fly and the results are visualized simultaneously. When a subset of samples and a phenotype variable is selected, a significance test on means (t-test or ranked based test; when phenotype variable is quantitative) or test of independence (chi-square or fisher’s exact test; when phenotype data is categorical) will be performed to test the association between the phenotype of interest with the selected samples. Additionally, other analyses can be easily added as extra shiny modules. Therefore, omicsViewer will greatly facilitate data exploration, many different hypotheses can be explored in a short time without the need for knowledge of R. In addition, the resulting data could be easily shared using a shiny server. Otherwise, a standalone version of omicsViewer together with designated omics data could be easily created by integrating it with portable R, which can be shared with collaborators or submitted as supplementary data together with a manuscript.

Depends R (>= 4.2)

License GPL-2

Imports survminer, survival, fastmatch, reshape2, stringr, beeswarm, grDevices, DT, shiny, shinythemes, shinyWidgets, plotly, networkD3, httr, matrixStats, RColorBrewer, Biobase, fgsea, openxlsx, psych, shinybusy, ggseqlogo, htmlwidgets, graphics, grid, stats, utils, methods, shinyjs, curl, flatxml, ggplot2, S4Vectors, SummarizedExperiment, RSQLite, Matrix, shinycssloaders

Suggests BiocStyle, knitr, rmarkdown, unittest

VignetteBuilder knitr

LazyData false

Encoding UTF-8

biocViews Software, Visualization, GeneSetEnrichment, DifferentialExpression, MotifDiscovery, Network, NetworkEnrichment

BugReports `https://github.com/mengchen18/omicsViewer`

URL `https://github.com/mengchen18/omicsViewer`

Video https://www.youtube.com/watch?v=0nirB-exquY&list=PLo2m88lJf-RRoLKMY8UEGqCpraKYrX5lk

RoxygenNote 7.1.2

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

git_branch RELEASE_3_15

git_last_commit 4651c9a

git_last_commit_date 2022-05-09

Date/Publication 2022-10-11

Author Chen Meng [aut, cre]

Maintainer Chen Meng <`mengchen18@gmail.com`>

R topics documented:

`app_module` *Application level 0 module*

Description

Function should only be used for the developers

Usage

```
app_module(
  input,
  output,
  session,
  .dir,
  filePattern = ".(RDS|db|sqlite|sqlite3)$",
  additionalTabs = NULL,
  ESVObj = reactive(NULL),
  esetLoader = readESVObj,
  exprsGetter = getExprs,
  pDataGetter = getPData,
  fDataGetter = getFData,
  imputeGetter = getExprsImpute,
  defaultAxisGetter = getAx,
```

```
  appName = "omicsViewer",
  appVersion = packageVersion("omicsViewer")
)
```

Arguments

`input`	input
`output`	output
`session`	session
`.dir`	reactive; directory containing the .RDS file of `ExpressionSet` or `SummarizedExperiment`
`filePattern`	file pattern to be displayed.
`additionalTabs`	additional tabs added to "Analyst" panel
`ESVObj`	the ESV object given, the drop down list should be disable in the "ui" component.
`esetLoader`	function to load the eset object, if an RDS file, should be "readRDS"
`exprsGetter`	function to get the expression matrix from eset
`pDataGetter`	function to get the phenotype data from eset
`fDataGetter`	function to get the feature data from eset
`imputeGetter`	function to get the imputed expression matrix from eset, only used when exporting imputed data to excel
`defaultAxisGetter`	function to get the default axes to be visualized. It should be a function with two arguments: x - the object loaded to the viewer; what - one of "sx", "sy", "fx" and "fy", representing the sample space x-axis, sample space y-axis, feature space x-axis and feature space y-axis respectively.
`appName`	name of the application
`appVersion`	version of the application

Value

do not return any values

Examples

```
if (interactive()) {
  dir <- system.file("extdata", package = "omicsViewer")
  server <- function(input, output, session) {
    callModule(app_module, id = "app", dir = reactive(dir))
  }
  ui <- fluidPage(
    app_ui("app")
  )
  shinyApp(ui = ui, server = server)
}
```

`app_ui` *Application level 0 UI*

Description

Function should only be used for the developers

Usage

```
app_ui(id, showDropList = TRUE, activeTab = "Feature")
```

Arguments

`id`	id
`showDropList`	logical; whether to show the dropdown list to select RDS file, if the ESVObj is given, this should be set to "FALSE"
`activeTab`	one of "Feature", "Feature table", "Sample", "Sample table", "Heatmap"

Value

a list of UI components

Examples

```
if (interactive()) {
  dir <- system.file("extdata", package = "omicsViewer")
  server <- function(input, output, session) {
    callModule(app_module, id = "app", dir = reactive(dir))
  }
  ui <- fluidPage(
    app_ui("app")
  )
  shinyApp(ui = ui, server = server)
}
```

`asEsetWithAttr` *Convert SummarizedExperiment to ExpressionSet retaining all attributes*

Description

Convert SummarizedExperiment to ExpressionSet retaining all attributes

Usage

```
asEsetWithAttr(x)
```

Arguments

`x`	an object of class SummarizedExperiment

Value

an object of class ExpressionSet

`correlationAnalysis` *Correlating a expression matrix with phenotypical variables*

Description

This is a convenience function to perform correlation analysis, the output is in a format ready to be incorporated into object to be visualized by `omicsViewer`.

Usage

```
correlationAnalysis(x, pheno, min.value = 12, prefix = "Cor")
```

Arguments

`x`	an expression matrix, rows are the features (e.g. proteins), columns are the samples
`pheno`	a `data.frame` storing the numerical phenotypical variable to be correlated with the rows (features) in expression matrix.
`min.value`	the minimum number of samples required in the correlation analysis, if lower than this number, NA will be returned.
`prefix`	prefix of the names. Usually don't need to be changed by the user. When changes are needed, the prefix should be in a format like [analysis name]|[subset] so the "analysis name" and "subset" can be selected in the `omicsViewer`.

Value

Every correlation analysis returns a `data.frame` with five columns: `R` - pearson correlation coefficient `N` - number of values used in the analysis `P` - p-values returned by pearson correlation analysis `logP` - log transformed p-values `range` - the range of values in expression matrix used in the analysis

Examples

```
e1 <- matrix(rnorm(500), 50, 10)
rownames(e1) <- paste0("FT", 1:50)
p1 <- matrix(rnorm(50), 10, 5)
colnames(p1) <- paste0("PH", 1:5)
colnames(e1) <- rownames(p1) <- paste0("S", 1:10)
correlationAnalysis(x = e1, pheno = p1, min.value = 8)
```

`csc2list` *convert a column compressed sparse matrix to a list*

Description

convert a column compressed sparse matrix to a list

Usage

```
csc2list(x)
```

Arguments

`x`	a matrix or CsparseMatrix object

Value

a sparse frame in data.frame

`exprspca` *Perform PCA and prepare results for* `omicsViewer`

Description

This is a convenience function to perform PCA on expression matrix, the output of PCA will be in a format ready to be incorporated into object to be visualized by `omicsViewer`.

Usage

```
exprspca(x, n = min(8, ncol(x) - 1), prefix = "PCA|All", fillNA = FALSE, ...)
```

Arguments

`x`	an expression matrix, where rows are features and samples are on columns.
`n`	number of components to keep
`prefix`	prefix of the names. Usually don't need to be changed by the user. When changes are needed, the prefix should be in a format like [analysis name]|[subset] so the "analysis name" and "subset" can be selected in the `omicsViewer`.
`fillNA`	logical; whether NA should be filled? If FALSE (default), `na.omit` will be called before PCA. If TRUE, the missing value will be replaced using `fillNA`.
`...`	other parameters passed to `prcomp`

Value

a `data.frame` storing the PCA results

Examples

```
# reading expression
packdir <- system.file("extdata", package = "omicsViewer")
expr <- read.delim(file.path(packdir, "expressionMatrix.tsv"), stringsAsFactors = FALSE)
# call PCA
pc <- exprspca(expr)
head(pc$samples)
head(pc$features)
```

`extendMetaData` *Add extra columns to the phenoData/colData or featureData/rowData in ExpressionSet/SummarizedExperiment*

Description

Add extra columns to the phenoData/colData or featureData/rowData in ExpressionSet/SummarizedExperiment

Add extra columns to the phenoData/colData or featureData/rowData in ExpressionSet/SummarizedExperiment

Add extra columns to the phenoData/colData or featureData/rowData in ExpressionSet/SummarizedExperiment

Usage

```
extendMetaData(object, newData, where)

## S4 method for signature 'ExpressionSet,data.frame'
extendMetaData(
  object,
  newData,
  where = c("pData", "fData", "colData", "rowData")[1]
)

## S4 method for signature 'SummarizedExperiment,data.frame'
extendMetaData(
  object,
  newData,
  where = c("pData", "fData", "colData", "rowData")[1]
)

## S4 method for signature 'SummarizedExperiment,DFrame'
extendMetaData(
  object,
  newData,
  where = c("pData", "fData", "colData", "rowData")[1]
)
```

Arguments

`object`	an object of `ExpressionSet-class`
`newData`	a `data.frame` containing the data to be added
`where`	where to add the extra columns, should be one of "pDataâ, "fData", "rowData" and "colData".

Value

an object of `ExpressionSet-class`

Note

The attributes in the pheno data and feature data will be preserved

Examples

```
est <- Biobase::ExpressionSet(assayData=matrix(runif(1000), nrow=100, ncol=10))
Biobase::pData(est)
est <- extendMetaData(est, data.frame(letter = letters[1:10]), where = "pData")
Biobase::pData(est)
```

`fgsea1` *Wrapper of fgseaMultilevel function to take binary gene set matrix as input*

Description

Wrapper of fgseaMultilevel function to take binary gene set matrix as input

Usage

```
fgsea1(gs, stats, gs_desc = NULL, ...)
```

Arguments

`gs`	either a data.frame or a (sparse) matrix input. If a data.frame object is given, it should have at least three columns named as "featureId", "gsId" and "weight". If a matrix is given, the matrix is binary matrix where rows are features and columns are gene sets. The values in the matrix should be either 1 or 0 representing the presence and absence of a feature in the genesets, repectively.
`stats`	ranking stats
`gs_desc`	description of gene sets, it should be a named vector and the names should be the same as colnames(gs)
`...`	other parameters passed to fgseaMultilevel

Value

a `data.frame` of fgsea results

Examples

```
## not for users
# library(fgsea)
# library(Biobase)
# dat <- readRDS(system.file(package = "omicsViewer", "extdata/demo.RDS"))
# fd <- fData(dat)
# fdgs <- fd[, grep("^GS\|", colnames(fd))]
# res <- fgsea1(fdgs, stats = fd$`t-test|OV_BR|md`, minSize = 5, maxSize = 500)
# res <- fgsea1(
#   fdgs, stats = fd$`t-test|OV_BR|md`,
#   minSize = 5, maxSize = 500, gs_desc = colnames(fdgs))
```

`fillNA` *Filling NAs in a matrix using constants calculated from user the defined function*

Description

This function is usually use to impute missing values in expression matrix, where the rows are feature and columns are samples. This function impute the missing values on the row-wise, that is, every row will be imputed using different constant.

Usage

```
fillNA(
  x,
  maxfill = quantile(x, probs = 0.15, na.rm = TRUE),
  fillingFun = function(x) min(x, na.rm = TRUE) - log10(2)
)
```

Arguments

`x`	a matrix with NA values
`maxfill`	the maximum filled value, if the value calculated by `fillingFun` is greater than `maxfill`, then `maxfill` will the used to replace NAs.
`fillingFun`	function to calculate the filling values. It should be a function accept at least one argument "x", which is a row of input expression matrix. The default is `function(x) min(x, na.rm = TRUE) - log10(2)` corresponds to the "half of lowest detected values" if the expression matrix is log10 transformed. More examples:#' `function(x) min(x, na.rm = TRUE) - 1` # half of lowest detected value when expression matrix is in log2 scale `function(x) 0` # replace NA by 0

Value

a matrix without NAs

Note

The returned matrix may have -Inf, which may need to be filtered/replaced additionally

Examples

```
m <- matrix(rnorm(200), 20, 10)
m[sample(1:200, size = 20)] <- NA
mf <- fillNA(m)
```

`filterRow` *Filter out rows of expression matrix*

Description

The function is used to filter rows with values of low intensities or do not reproducible presented in replicates.

Usage

```
filterRow(x, max.quantile = NULL, max.value = NULL, var = NULL, min.rep = 2)
```

Arguments

`x`	an expression matrix
`max.quantile`	a single numerical value between (0, 1), if the row maximum is smaller than this quantile (calculated from the whole matrix), the row will be removed.
`max.value`	a single numerical value, if the the maximum value of a rwo is smaller than this value, the row will be removed. Only used if `max.quantile` is set to "NULL".
`var`	variables has the same length as the column number in `x` to indicate which sample is from which group
`min.rep`	the minimum number of replicate in at least one of the groups, if less than this value, the row will be removed.

Value

a logical vector where the TRUE means row to keep

Examples

```
e1 <- matrix(rnorm(5000, sd = 0.3), 500, 10) + rnorm(500)
f <- filterRow(x = e1, max.quantile = 0.25)
table(f)
```

`getAutoRIF` *Get genes associated with search terms and AutoRIF annotations*

Description

Get genes associated with search terms and AutoRIF annotations

Usage

```
getAutoRIF(term, rif = c("generif", "autorif")[1], filter = TRUE)
```

Arguments

`term`	a character vector of terms want to search
`rif`	either autorif or generif, see "https://maayanlab.cloud/geneshot/"
`filter`	whether the result should be filtered. The least frequently mentioned genes (most like 1 or 2 times) will be removed.

Value

a `data.frame` of 4 columns: gene, n, perc, rank.

Note

https://amp.pharm.mssm.edu/geneshot/

References

Alexander Lachmann, Brian M Schilder, Megan L Wojciechowicz, Denis Torre, Maxim V Kuleshov, Alexandra B Keenan, Avi Ma'ayan, Geneshot: search engine for ranking genes from arbitrary text queries, Nucleic Acids Research, Volume 47, Issue W1, 02 July 2019, Pages W571–W577, https://doi.org/10.1093/nar/gkz393

Alexander Lachmann, Brian M Schilder, Megan L Wojciechowicz, Denis Torre, Maxim V Kuleshov, Alexandra B Keenan, Avi Ma'ayan, Geneshot: search engine for ranking genes from arbitrary text queries, Nucleic Acids Research, Volume 47, Issue W1, 02 July 2019, Pages W571–W577, https://doi.org/10.1093/nar/gkz393

Examples

```
a <- getAutoRIF("mtor signaling")
```

`getMQParams` *Parse mqpar.xml file*

Description

Getting the experimental informatione (TMT or label free) from mqpar.xml file.

Usage

```
getMQParams(x)
```

Arguments

`x`	the path to mqpar.xml file

Value

a list of MQ paramters

`getUPRefProteomeID` *get uniprot reference proteome IDs*

Description

get uniprot reference proteome IDs

get uniprot reference proteome IDs

Usage

```
getUPRefProteomeID(
  domain = c("Eukaryota", "Archaea", "Bacteria", "Viruses")[1]
)

downloadUPRefProteome(
  id,
  domain = c("Eukaryota", "Archaea", "Bacteria", "Viruses")[1],
  destdir = "./"
)
```

Arguments

`domain`	the domain, one of "Eukaryota", "Archaea", "Bacteria" or "Viruses"
`id`	the UP id to download
`destdir`	destination directory

Value

a character vector of UP ids

a character vector of UP ids

Functions

- `getUPRefProteomeID`: get uniprot reference protein IDs

`gsAnnotIdList` *Annotation of gene/protein function using multiple IDs.*

Description

Annotation of gene/protein function using multiple IDs.

Usage

```
gsAnnotIdList(
  idList,
  gsIdMap,
  minSize = 5,
  maxSize = 500,
  data.frame = FALSE,
  sparse = TRUE
)
```

Arguments

`idList`	list of protein IDs, e.g. list(c("ID1", "ID2"), c("ID13"), c("ID4", "ID8", "ID10"))
`gsIdMap`	a data frame for geneset to id map, it has two columns - id: the ID column - term: annotation terms e.g. gsIdMap <- data.frame(id = c("ID1", "ID2", "ID1", "ID2", "ID8", "ID10"), term = c("T1", "T1", "T2", "T2", "T2", "T2"), stringsAsFactors = FALSE)
`minSize`	minimum size of gene sets
`maxSize`	maximum size of gene sets
`data.frame`	logical; whether to organize the result into `data.frame` format, see "Value" section.
`sparse`	logical; whether to return a sparse matrix, only used when data.frame=FALSE

Value

A binary matrix (if `data.frame` = FALSE), the number of rows is the same with length of idList, the columns are the annotated gene set; or a `data.frame` (if `data.frame` = TRUE) with three columns: featureId, gsId, weight.

Examples

```
terms <- data.frame(
  id = c("ID1", "ID2", "ID1", "ID2", "ID8", "ID10"),
 term = c("T1", "T1", "T2", "T2", "T2", "T2"),
  stringsAsFactors = FALSE
)
features <- list(c("ID1", "ID2"), c("ID13"), c("ID4", "ID8", "ID10"))
gsAnnotIdList(idList = features, gsIdMap = terms, minSize = 1, maxSize = 500)

terms <- data.frame(
id = c("ID1", "ID2", "ID1", "ID2", "ID8", "ID10", "ID4", "ID4"),
term = c("T1", "T1", "T2", "T2", "T2", "T2", "T1", "T2"),
stringsAsFactors = FALSE
)
features <- list(F1 = c("ID1", "ID2", "ID4"), F2 = c("ID13"), F3 = c("ID4", "ID8", "ID10"))
gsAnnotIdList(features, gsIdMap = terms, data.frame = TRUE, minSize = 1)
gsAnnotIdList(features, gsIdMap = terms, data.frame = FALSE, minSize = 1)
```

`hasAttr` *Check whether an object has an attribute*

Description

Check whether an object has an attribute

Usage

```
hasAttr(x, attr.name)
```

Arguments

`x`	the object
`attr.name`	a character vector containing the name of attributes to be checked

Value

a logical value/vector has the same length as attr.name

`hclust2str` *Convert hclust object to/from single character*

Description

Convert hclust object to/from single character

Usage

```
hclust2str(x)

str2hclust(x)
```

Arguments

`x`	a character of length one or an hclust object

Value

a character stores the hclust object

a hclust object

Note

The $call element in hclust will not retained in the conversion. The conversion decrease the precision in $height element.

Examples

```
# not for end users
# m <- matrix(rnorm(50), 25)
# hc <- hclust(dist(m))
# plot(hc)
# te <- hclust2str(hc)
# hc2 <- str2hclust(te)
# plot(hc2)
```

`list2csc` *convert a list to column compressed sparse matrix*

Description

convert a list to column compressed sparse matrix

Usage

```
list2csc(l, dimnames)
```

Arguments

`l`	a data.frame with at least two columns - featureId, gsId; optionally a "weight" column.
`dimnames`	a list of dimnames, should contain at least one element for the row names.

Value

a sparse matrix, CsparseMatrix, column compressed

`multi.t.test` *Function to perform multiple t-tests on an expression matrix*

Description

This is a convenience function to perform multiple student's t-test. The output is in a format ready to be incorporated into object to be visualized by `omicsViewer`. This function use `t.test`.

Usage

```
multi.t.test(x, pheno, compare = NULL, fillNA = FALSE, ...)
```

Arguments

`x`	an expression matrix, usually log10 transformed.
`pheno`	phenotype data of `x`, the number of rows in `pheno` must equal the number of columns of `x`. Please refer to examples for more details.
`compare`	NULL or a matrix with three columns to define the comparisons to do. When a matrix is given, the first column should be one of the column headers in `pheno`; then the second and third columns should be two values presented (more than once) in the columns of `pheno` selected by the values in the first column. The samples mapped to the two values are compared. If paired comparisons to be done, the orders of samples should be mapped

`fillNA`	logical; whether NA should be filled? If FALSE (default), t test will be performed whenever possible. If not possible, then NA will be returned. If TRUE, the missing value will be replaced using `fillNA`.
`...`	other parameters passed to `t.test`

Value

a `data.frame` stores the t-test results with the follow columns: `mean|[selected header in pheno]|[group 1 in test]` - The mean value of group 1 `n value|[selected header in pheno]|[group 1 in test]` - The number of value used in the test for group 1 `quantile|[selected header in pheno]|[group 1 in test]` - The quantile of means values in group 1 `mean|[selected header in pheno]|[group 2 in test]` - The mean value of group 2 `n value|[selected header in pheno]|[group 2 in test]` - The number of value used in the test for group 2 `quantile|[selected header in pheno]|[group 2 in test]` - The quantile of means values in group 2 `ttest|[group 1 in test]_vs_[group 2 in test]|pvalue` - The p-value return by `t.test` `ttest|[group 1 in test]_vs_[group 2 in test]|log.pvalue` - The -log10 transformed p-value `ttest|[group 1 in test]_vs_[group 2 in test]|fdr` - The BH method corrected p-values, e.g. FDR `ttest|[group 1 in test]_vs_[group 2 in test]|log.fdr` - The -log10 transformed FDR `ttest|[group 1 in test]_vs_[group 2 in test]|mean.diff` - The difference between the means of the two groups, e.g. fold change

Examples

```
# reading expression
packdir <- system.file("extdata", package = "omicsViewer")
expr <- read.delim(file.path(packdir, "expressionMatrix.tsv"), stringsAsFactors = FALSE)
# reading phenotype data
pd <- read.delim(file.path(packdir, "sampleGeneral.tsv"), stringsAsFactors = FALSE)

## Single t-test
head(pd)
# define comparisons
tests <- c("Origin", "RE", "ME")
tres <- multi.t.test(x = expr, pheno = pd, compare = tests)

## multiple t-test
head(pd)
# define comparisons
tests <- rbind(
c("Origin", "RE", "ME"),
c("Origin", "RE", "LE"),
c('TP53.Status', "MT", "WT")
)
tres <- multi.t.test(x = expr, pheno = pd, compare = tests)
```

`nColors` *Generating k distinct colors*

Description

Mainly used in the shiny app to generate reproducible k distinct colors.

Usage

```
nColors(k, stop = FALSE)
```

Arguments

`k`	a number between 1 to 60 tells how many distinct colors to use
`stop`	logical; whether the function should return an error message if k is not in the range of 2 to 60. Default FALSE, the function will return NULL.

Value

a vector of hex code for k colors or NULL

Examples

```
nColors(5)
nColors(1, stop = FALSE)
```

`normalize.nQuantiles` *Normalization using n quantiles*

Description

Normalization using n quantiles

Usage

```
normalize.nQuantiles(x, probs = 0.5, shareFeature = FALSE, ref = 1)
```

Arguments

`x`	an expression matrix, usually log transformed
`probs`	the quantiles to be aligned across samples. If `probs` is a length 1 numerical vector, the quantiles will aligned. As a special case, `probs = 0.5` equals the median centering. If `probs`' length is > 1, a shift and scaling factor of samples will be calculating by fitting linear models using quantiles of samples, the median and variance of samples will be corrected using the intersect and slope of the fitted model.
`shareFeature`	logocal; if TRUE, the normalization will be based on the shared features between samples
`ref`	the columns name or index to specify the reference sample, only used when `shareFeature = TRUE`

Value

a normalized matrix

Examples

```
e1 <- matrix(rnorm(5000), 500, 10)
e1[, 6:10] <- 0.3 *e1[, 6:10] + 3
boxplot(e1)
# median centering, no variance correction
e2 <- normalize.nQuantiles(x = e1, probs = 0.5)
boxplot(e2)
# median centering + variance stablization
e3 <- normalize.nQuantiles(x = e1, probs = seq(0.25, 0.75, by = 0.1))
boxplot(e3)
```

`normalize.totsum` *Normalize total sum*

Description

Normalize total sum

Usage

```
normalize.totsum(x)
```

Arguments

`x`	a log10 transformed expression matrix

Value

a normalized matrix

Examples

```
e1 <- matrix(rnorm(5000), 500, 10)
e1[, 6:10] <- e1[, 6:10]+3
boxplot(e1)
e2 <- normalize.totsum(x = e1)
boxplot(e2)
```

`normalizeColWise` *Column-wise normalization of expression matrix*

Description

A wrapper function of all column-wise normalization methods

Usage

```
normalizeColWise(
  x,
  method = c("Median centering", "Median centering (shared ID)", "Total sum",
    "median centering + variance stablization")[1]
)
```

Arguments

`x`	an expression matrix where rows are features and columns are samples, usually log transformed.
`method`	normalization method to use "Median centering" - median centering, see `normalize.nQuantiles` "Median centering (shared ID)" - median centering using shared features, see `normalize.nQuantiles` "Total sum" - total sum normalization "median centering + variance stablization" - 10 quantile normalization using 0.25, 0.3, ..., 0.75, see `normalize.nQuantiles`

Value

a normalized matrix

Examples

```
e1 <- matrix(rnorm(5000), 100, 50)+10
boxplot(e1)
e2 <- normalizeColWise(x = e1, method = "Median centering")
boxplot(e2)
```

`normalizeData` *Normalized expression matrix*

Description

A wrapper function of all normalization methods, including row-wise or column-wise normalization.

Usage

```
normalizeData(
  x,
 colWise = c("None", "Median centering", "Median centering (shared ID)", "Total sum",
    "median centering + variance stablization")[1],
  rowWise = c("None", "Reference", "Batch mean", "Batch reference")[1],
  ref = NULL,
  batch = NULL
)
```

Arguments

`x`	an expression matrix where rows are features and columns are samples, usually log transformed.
`colWise`	column-wise normalization method to use, see `normalizeColWise`
`rowWise`	row-wise normalization method to used `Reference` - using `removeVarQC` method `Batch mean` - using `rowshift` method without reference samples `Batch reference` - using `rowshift` method with reference samples
`ref`	index of reference samples
`batch`	batch factor

Value

a normalized matrix

Examples

```
e1 <- matrix(rnorm(5000), 100, 50)+10
boxplot(e1)
e2 <- normalizeData(x = e1, ref = seq(5, 45, by = 10), rowWise = "Reference")
boxplot(e2)
```

`omicsViewer` *Start omicsViewer*

Description

Start omicsViewer

Usage

```
omicsViewer(
  dir,
  additionalTabs = NULL,
  filePattern = ".(RDS|DB|SQLITE|SQLITE3)$",
```

```
  ESVObj = NULL,
  esetLoader = readESVObj,
  exprsGetter = getExprs,
  pDataGetter = getPData,
  fDataGetter = getFData,
  defaultAxisGetter = getAx,
  appName = "omicsViewer",
  appVersion = packageVersion("omicsViewer")
)
```

Arguments

`dir`	directory to the `ExpressionSet` or `SummarizedExperiment` object. Only give the directory in this argument, not the .rds file.
`additionalTabs`	additional tabs added to "Analyst" panel
`filePattern`	file pattern to be displayed.
`ESVObj`	the ESV object
`esetLoader`	function to load the eset object, if an RDS file, should be "readRDS"
`exprsGetter`	function to get the expression matrix from eset
`pDataGetter`	function to get the phenotype data from eset
`fDataGetter`	function to get the feature data from eset
`defaultAxisGetter`	function to get the default axes to be visualized. It should be a function with two arguments: x - the object loaded to the viewer; what - one of "sx", "sy", "fx" and "fy", representing the sample space x-axis, sample space y-axis, feature space x-axis and feature space y-axis respectively.
`appName`	name of the application
`appVersion`	version of the application

Value

do not return values

Examples

```
1
## To start the shiny app:
# omicsViewer(
#  system.file("extdata", package = "omicsViewer")
# )
```

`parseDatTerm` *Extract function annotation from uniprot .dat file*

Description

Extract function annotation from uniprot .dat file

Usage

```
parseDatTerm(file, outputDir = NULL, ...)
```

Arguments

`file`	the .dat or .dat.gz file
`outputDir`	dir of output file
`...`	other parameters passed to readLines

Value

a data.frame parse from .dat file

`plotly_boxplot_module` *Shiny module for boxplot using plotly - Module*

Description

Shiny module for boxplot using plotly - Module

Usage

```
plotly_boxplot_module(
  input,
  output,
  session,
  reactive_param_plotly_boxplot,
  reactive_checkpoint = reactive(TRUE)
)
```

Arguments

`input`	input
`output`	output
`session`	session
`reactive_param_plotly_boxplot`	reactive value; argument passed to plotly_boxplot
`reactive_checkpoint`	reactive_value; check this value before render any plot/executing any calculation

Value

do not return any values

Examples

```
if (interactive()) {

  library(shiny)

  ui <- fluidPage(
    plotly_boxplot_ui("testplotly")
  )

  server <- function(input, output, session) {

    x <- cbind(matrix(rnorm(10000, mean = 3), 1000, 10), matrix(rnorm(20000), 1000, 20))
    x[sample(1:length(x), size = 0.3*length(x))] <- NA
    rownames(x) <- paste("R", 1:nrow(x), sep = "")
    colnames(x) <- paste("C", 1:ncol(x), sep = "")
    callModule(plotly_boxplot_module, id = "testplotly",
               reactive_param_plotly_boxplot = reactive(list(
                 x = x# , i  = c(4, 20, 80)# , highlight = c(1, 4, 5, 20), extvar = 1:30
               ))
    )
  }

  shinyApp(ui, server)
}
```

`plotly_boxplot_ui` *Shiny module for boxplot using plotly - UI*

Description

Function should only be used for the developers

Usage

```
plotly_boxplot_ui(id)
```

Arguments

`id`	id

Value

a tagList of UI components

a tagList of UI components

Examples

```
if (interactive()) {

  library(shiny)

  ui <- fluidPage(
    plotly_boxplot_ui("testplotly")
  )

  server <- function(input, output, session) {

    x <- cbind(matrix(rnorm(10000, mean = 3), 1000, 10), matrix(rnorm(20000), 1000, 20))
    x[sample(1:length(x), size = 0.3*length(x))] <- NA
    rownames(x) <- paste("R", 1:nrow(x), sep = "")
    colnames(x) <- paste("C", 1:ncol(x), sep = "")
    callModule(plotly_boxplot_module, id = "testplotly",
               reactive_param_plotly_boxplot = reactive(list(
                 x = x# , i  = c(4, 20, 80)# , highlight = c(1, 4, 5, 20), extvar = 1:30
               ))
    )
  }

  shinyApp(ui, server)
}
```

`plotly_scatter_module` *Shiny module for scatter plot using plotly - Module*

Description

Function should only be used for the developers

Usage

```
plotly_scatter_module(
  input,
  output,
  session,
  reactive_param_plotly_scatter,
  reactive_regLine = reactive(FALSE),
  reactive_checkpoint = reactive(TRUE),
  htest_var1 = reactive(NULL),
  htest_var2 = reactive(NULL)
)
```

Arguments

`input`	input

`output`	output
`session`	sesion
`reactive_param_plotly_scatter`	reactive parammeters for plotly_scatter
`reactive_regLine`	logical show or hide the regression line
`reactive_checkpoint`	checkpoint
`htest_var1`	when the plot is a beeswarmplot, two groups could be selected for two group comparison, this argument gives the default value. Mainly used for restoring the saved session.
`htest_var2`	see above

Value

a list containing the information about the selected data points

an reactive object containing the information of selected, brushed points.

Examples

```
if (interactive()) {
  library(shiny)

  # two random variables
  x <- rnorm(30)
  y <- x + rnorm(30, sd = 0.5)

  # variables mapped to color, shape and size
  cc <- sample(letters[1:4], replace = TRUE, size = 30)
  shape <- sample(c("S1", "S2", "S3"), replace = TRUE, size = 30)
  sz <- sample(c(10, 20, 30, replace = TRUE, size = 30))

  ui <- fluidPage(
    plotly_scatter_ui("test_scatter")
  )

  server <- function(input, output, session) {
    v <- callModule(plotly_scatter_module, id = "test_scatter",
                    # reactive_checkpoint = reactive(FALSE),
                    reactive_param_plotly_scatter = reactive(list(
                      x = x, y = y,
                      color = cc,
                      shape = shape,
                      size = sz,
                      tooltips = paste("A", 1:30)
                    )))
    observe(print(v()))
  }
  shinyApp(ui, server)
```

```
  # example beeswarm horizontal
  x <- rnorm(30)
  y <- sample(c("x", "y", "z"), size = 30, replace = TRUE)
  shinyApp(ui, server)

  # example beeswarm vertical
  x <- sample(c("x", "y", "z"), size = 30, replace = TRUE)
  y <- rnorm(30)
  shinyApp(ui, server)

  # return values
  x <- c(5, 6, 3, 4, 1, 2)
  y <- c(5, 6, 3, 4, 1, 2)
  ui <- fluidPage(
    plotly_scatter_ui("test_scatter")
  )
  server <- function(input, output, session) {
    v <- callModule(plotly_scatter_module, id = "test_scatter",
                    reactive_param_plotly_scatter = reactive(list(
                      x = x, y = y, tooltips = paste("A", 1:6), highlight = 2:4
                    )))

    observe(print(v()))
  }
  shinyApp(ui, server)
}
```

`plotly_scatter_ui` *Shiny module for scatter plot using plotly - UI*

Description

Function should only be used for the developers

Usage

```
plotly_scatter_ui(id, height = "400px")
```

Arguments

`id`	id
`height`	figure height

Value

a tagList of UI components

Examples

```
if (interactive()) {
  library(shiny)

  # two random variables
  x <- rnorm(30)
  y <- x + rnorm(30, sd = 0.5)

  # variables mapped to color, shape and size
  cc <- sample(letters[1:4], replace = TRUE, size = 30)
  shape <- sample(c("S1", "S2", "S3"), replace = TRUE, size = 30)
  sz <- sample(c(10, 20, 30, replace = TRUE, size = 30))

  ui <- fluidPage(
    plotly_scatter_ui("test_scatter")
  )

  server <- function(input, output, session) {
    v <- callModule(plotly_scatter_module, id = "test_scatter",
                    # reactive_checkpoint = reactive(FALSE),
                    reactive_param_plotly_scatter = reactive(list(
                      x = x, y = y,
                      color = cc,
                      shape = shape,
                      size = sz,
                      tooltips = paste("A", 1:30)
                    )))
    observe(print(v()))
  }
  shinyApp(ui, server)



  # example beeswarm horizontal
  x <- rnorm(30)
  y <- sample(c("x", "y", "z"), size = 30, replace = TRUE)
  shinyApp(ui, server)

  # example beeswarm vertical
  x <- sample(c("x", "y", "z"), size = 30, replace = TRUE)
  y <- rnorm(30)
  shinyApp(ui, server)

  # return values
  x <- c(5, 6, 3, 4, 1, 2)
  y <- c(5, 6, 3, 4, 1, 2)
  ui <- fluidPage(
    plotly_scatter_ui("test_scatter")
  )
  server <- function(input, output, session) {
    v <- callModule(plotly_scatter_module, id = "test_scatter",
                    reactive_param_plotly_scatter = reactive(list(
```

```
                        x = x, y = y, tooltips = paste("A", 1:6), highlight = 2:4
                      )))

    observe(print(v()))
  }
  shinyApp(ui, server)
}
```

`prepOmicsViewer` *Prepare object to be viewed by omicsViewer*

Description

This is a convenience function to prepare the data to be visualized using `omicsViewer`. The result of PCA and t-test could be included directly.

Usage

```
prepOmicsViewer(
  expr,
  pData,
  fData,
  PCA = TRUE,
  ncomp = min(8, ncol(expr)),
  pca.fillNA = TRUE,
  t.test = NULL,
  ttest.fillNA = FALSE,
  ...,
  gs = NULL,
  stringDB = NULL,
  surv = NULL,
  SummarizedExperiment = TRUE
)
```

Arguments

`expr`	expression matrix where the rows are feature and columns are samples, matrix should be log10 transformed and have unique row and column names
`pData`	phenotype data
`fData`	feature data
`PCA`	pca
`ncomp`	number of components to keep
`pca.fillNA`	logical, whether the NA should be filled with a constant in PCA.
`t.test`	will be passed to the `compare` argument in `multi.t.test`
`ttest.fillNA`	logical, whether the NA should be filled with a constant in t-test.

`...`	arguments passed to `t.test`, such as `paired`.
`gs`	gene-set data, please refer to examples for more details about the format
`stringDB`	the IDs that can be used in the STRING database (https://string-db.org/) query.
`surv`	survival data, please refer to examples for more details about the format
`SummarizedExperiment`	logical; whether to return an object of class `SummarizedExperiment`. If set to FALSE, the function will return an `ExpressionSet` object.

Value

an object of `ExpressionSet` or `SummarizedExperiment` that can be visualized using `omicsViewer`

Examples

```
packdir <- system.file("extdata", package = "omicsViewer")
# reading expression
expr <- read.delim(file.path(packdir, "expressionMatrix.tsv"), stringsAsFactors = FALSE)
colnames(expr) <- make.names(colnames(expr))
rownames(expr) <- make.names(rownames(expr))
# reading feature data
fd <- read.delim(file.path(packdir, "featureGeneral.tsv"), stringsAsFactors = FALSE)
# reading phenotype data
pd <- read.delim(file.path(packdir, "sampleGeneral.tsv"), stringsAsFactors = FALSE)

#  reading other datasets
drugData <- read.delim(file.path(packdir, "sampleDrug.tsv"))
# survival data
# this data is from cell line, the survival data are fake data to
# show how to use the survival data in #' omicsViewer
surv <- read.delim(file.path(packdir, "sampleSurv.tsv"))
# gene set information
genesets <- read_gmt(file.path(packdir, "geneset.gmt"), data.frame = TRUE)
gsannot <- gsAnnotIdList(idList = rownames(fd), gsIdMap = genesets, data.frame = TRUE)

# Define t-test to be done, a matrix nx3
# every row define a t-test, the format
# [column header] [group 1 in the test] [group 2 in the test]
tests <- rbind(
 c("Origin", "RE", "ME"),
 c("Origin", "RE", "LE"),
 c('TP53.Status', "MT", "WT")
 )
# prepare column for stringDB query
strid <- sapply(strsplit(fd$Protein.ID, ";|-"), "[", 1)
###
d <- prepOmicsViewer(
  expr = expr, pData = pd, fData = fd,
  PCA = TRUE, pca.fillNA = TRUE,
  t.test = tests, ttest.fillNA = FALSE,
  gs = gsannot, stringDB = strid, surv = surv)
# feature space - default x axis
```

```
attr(d, "fx") <- "ttest|RE_vs_ME|mean.diff"
# feature space - default y axis
attr(d, "fy") <- "ttest|RE_vs_ME|log.fdr"
# sample space - default x axis
attr(d, "sx") <- "PCA|All|PC1("
# sample space - default y axis
attr(d, "sy") <- "PCA|All|PC2("
# Save object and view
# saveRDS(d, file = "dtest.RDS")
##  to open the viewer
# omicsViewer("./")
```

`read.proteinGroups` *Reading proteinGroup table of MaxQuant output*

Description

A convenience function to read the proteinGroups table of MaxQuant output. The function organize the result into different tables, e.g. iBAQ.

Usage

```
read.proteinGroups(x, quant = c("LF", "TMT")[1])
```

Arguments

`x`	the proteinGroup.txt file returned by MaxQuant search
`quant`	the quantification method, LF or TMT

Value

a list of tables extracted from proteinGroups.txt file

`read.proteinGroups.lf` *Read protein groups output of maxquant output and split it to columns*

Description

Read protein groups output of maxquant output and split it to columns

Usage

```
read.proteinGroups.lf(file)
```

Arguments

`file`	Maxquant proteinGroup.txt file path

Value

a list of tables extracted from proteinGroups.txt file

`readESVObj` *Read the object of SummarizedExperiment or ExpressetSet to be visualized using omicsViewer*

Description

This function accept a path to a sqlite database or RDS object. If an RDS file to be read, The function is similar to `readRDS`. It reads the object to R working environment and perform extra two things. 1. If the loaded data an class of `SummarizedExperiment`, it will be converted to `ExpressionSet`; 2. If the gene set annotatio is in matrix format, the gene set annotation is converted to `data.frame` format.

Usage

```
readESVObj(x)
```

Arguments

`x`	the path of an object of `SummarizedExperiment` or `ExpressionSet`, passed to readRDS

Value

an object of class `ExpressionSet` or `SummarizedExperiment` to be visualzied.

Examples

```
file <- system.file("extdata/demo.RDS", package = "omicsViewer")
obj <- readESVObj(file)
```

`read_gmt` *Reading gene set .gmt file*

Description

Frequently the .gmt files are downloaed from MSigDB database

Usage

```
read_gmt(x, id = NA, data.frame = FALSE)
```

Arguments

`x`	the name/path of the gmt file to be read
`id`	the id used in gene sets, if is not NA, it should be either "SYMBOL" or "ENTREZ". Usually only used when reading the .gmt file downloaded from MSigDB.
`data.frame`	logical; whether to organize the data in `data.frame` format. Default is FALSE, a list will be returned.

Value

a list or data frame of gene set. When data.frame = TRUE, the returned object is a `data.frame` with two columns: id and term.

Examples

```
file <- system.file("extdata", package = "omicsViewer")
file <- file.path(file, "geneset.gmt")
gs <- read_gmt(file)
```

`removeVarQC` *Removing variance of reference samples*

Description

This normalization removes the variance in reference samples. The method do not need to specific the batch assignment but cannot work with data contains less than five common reference samples. A typical use of this normalization is to correct some drifting effect in mass spec based label free proteomics or untargeted metabolomics experiment. Usually, this is a very strong normalization should only be used with good reasons.

Usage

```
removeVarQC(x, ref, positive = TRUE, ...)
```

Arguments

`x`	an expression matrix
`ref`	the index of reference samples
`positive`	logical; force only positive values in the resulted matrix
`...`	if given, `normalize.nQuantiles` will be called first, the arguments here will be passed to `normalize.nQuantiles`

Value

a normalized matrix

Examples

```
e1 <- matrix(rnorm(5000), 100, 50)+10
e2 <- removeVarQC(x = e1, ref = seq(5, 45, by = 10))
boxplot(e2)
```

`rowshift` *Row-wise normalization of expression matrix with or without reference sample*

Description

Row-wise normalization of expression matrix with or without reference sample

Usage

```
rowshift(x, batch, ref = NULL, useMean = FALSE)
```

Arguments

`x`	an expression matrix where rows are features, e.g. genes, proteins and columns are samples. The values in the matrix are usually log transformed.
`batch`	a factor or vector has the same length as `ncol(x)` to indicate the batch assignment of samples.
`ref`	a logical vector has the same length as `ncol(x)` to indicated which columns are the common references among batches. If it is NULL (by default), the mean of all channels will be used as batch reference. When NA present in the reference channels, the mean values will be used in correction.
`useMean`	logical; whether to use means of batches, usually set to TRUE when no reference available

Value

a matrix (hopefully without/with less batch effect)

Examples

```
e1 <- matrix(rnorm(5000), 500, 10)
e1[, 6:10] <- e1[, 6:10] + 3
boxplot(e1)
f <- rep(c("a", "b"), each = 5)
e2 <- rowshift(x = e1, batch = f)
boxplot(e2)
```

`saveOmicsViewerDb` *Save the xcmsViewer result object as sqlite database*

Description

Save the xcmsViewer result object as sqlite database

Usage

```
saveOmicsViewerDb(obj, db.file, overwrite = TRUE)

## S4 method for signature 'SummarizedExperiment,character'
saveOmicsViewerDb(obj, db.file, overwrite = TRUE)

## S4 method for signature 'ExpressionSet,character'
saveOmicsViewerDb(obj, db.file, overwrite = TRUE)
```

Arguments

`obj`	an object of class ExpressionSet or SummarizedExperiment
`db.file`	a character indicate file name of the database file
`overwrite`	logical. whether the database should be overwritten if exist already.

Value

the directory where the database saved

Examples

```
f <- system.file("extdata", "demo.RDS", package = "omicsViewer")
es <- readRDS(f)
# The following line will write a database file on your disk
# saveOmicsViewerDb(es, db.file = "./omicsViewerData.db")
```

`triselector_module` *The three-step selector - the module function*

Description

The selector is used to select columns of phenotype and feature data. Function should only be used for the developers.

Usage

```
triselector_module(
  input,
  output,
  session,
  reactive_x,
  reactive_selector1 = reactive(NULL),
  reactive_selector2 = reactive(NULL),
  reactive_selector3 = reactive(NULL),
  label = "Group Label:"
)
```

Arguments

`input`	input
`output`	output
`session`	session
`reactive_x`	an nx3 matrix
`reactive_selector1`	default value for selector 1
`reactive_selector2`	default value for selector 2
`reactive_selector3`	default value for selector 3
`label`	of the triselector

Value

an reactive object containing the selected values

Examples

```
if (interactive()) {
  library(shiny)
  library(Biobase)

  file <- system.file("extdata/demo.RDS", package = "omicsViewer")
  dat <- readRDS(file)
  fData <- fData(dat)
  triset <- stringr::str_split_fixed(colnames(fData), '\\|', n= 3)

  ui <- fluidPage(
    triselector_ui("tres"),
    triselector_ui("tres2")
  )
  server <- function(input, output, session) {
    v1 <- callModule(triselector_module, id = "tres", reactive_x = reactive(triset),
                     reactive_selector1 = reactive("ttest"),
                     reactive_selector2 = reactive("RE_vs_ME"),
```

```
                     reactive_selector3 = reactive("mean.diff")
    )
    v2 <- callModule(triselector_module, id = "tres2", reactive_x = reactive(triset),
                     reactive_selector1 = reactive("ttest"),
                     reactive_selector2 = reactive("RE_vs_ME"),
                     reactive_selector3 = reactive("log.fdr"))
    observe({
      print("//////////////////////////")
      print(v1())
    })
  }

  shinyApp(ui, server)
}
```

`triselector_ui` *The three-step selector - the ui function*

Description

Function should only be used for the developers

Usage

```
triselector_ui(id)
```

Arguments

`id`	id

Value

a tagList of UI components

Examples

```
if (interactive()) {
  library(shiny)
  library(Biobase)

  file <- system.file("extdata/demo.RDS", package = "omicsViewer")
  dat <- readRDS(file)
  fData <- fData(dat)
  triset <- stringr::str_split_fixed(colnames(fData), '\\|', n= 3)

  ui <- fluidPage(
    triselector_ui("tres"),
    triselector_ui("tres2")
  )
  server <- function(input, output, session) {
```

```
    v1 <- callModule(triselector_module, id = "tres", reactive_x = reactive(triset),
                     reactive_selector1 = reactive("ttest"),
                     reactive_selector2 = reactive("RE_vs_ME"),
                     reactive_selector3 = reactive("mean.diff")
    )
    v2 <- callModule(triselector_module, id = "tres2", reactive_x = reactive(triset),
                     reactive_selector1 = reactive("ttest"),
                     reactive_selector2 = reactive("RE_vs_ME"),
                     reactive_selector3 = reactive("log.fdr"))
    observe({
      print("/////////////////////////")
      print(v1())
    })
  }

  shinyApp(ui, server)
}
```

`trisetter` *Create a nx3 matrix that can be use for triselector given a meta and expression table*

Description

only used inside reactive

Usage

```
trisetter(meta, expr = NULL, combine)
```

Arguments

`meta`	a meta data, usually either phenotype data or feature data
`expr`	expression matrix, optional.
`combine`	how the meta and expression to be combined. Should be either "pheno" or "feature' or "none".

Value

a nx3 matrix

a `data.frame` with 3 columns

`validMQFolder` *MQ folder validator Validate whether a folder is a MQ output folder*

Description

MQ folder validator Validate whether a folder is a MQ output folder

Usage

```
validMQFolder(dir)
```

Arguments

`dir`	the directory to check

Details

from the root level, these files exist: mqpar.xml [[combined/]txt/]proteinGroups.txt

Value

a list containing the info about MQ folder check

`varSelector` *variable selector*

Description

variable selector

Usage

```
varSelector(x, expr, meta, alternative = NULL)
```

Arguments

`x`	variable return by triselector, a list of length three named as "analysis", "subset" and "variable’
`expr`	the expression matrix
`meta`	a meta matrix
`alternative`	alternative value to be returned when nothing to select

Value

the selected values in input argument `x`

Index