Package ‘DEP’

May 14, 2024

Title Differential Enrichment analysis of Proteomics data

Version 1.26.0

Description This package provides an integrated analysis workflow for robust and reproducible analysis of mass spectrometry proteomics data for differential protein expression or differential enrichment. It requires tabular input (e.g. txt files) as generated by quantitative analysis softwares of raw mass spectrometry data, such as MaxQuant or IsobarQuant. Functions are provided for data preparation, filtering, variance normalization and imputation of missing values, as well as statistical testing of differentially enriched / expressed proteins. It also includes tools to check intermediate steps in the workflow, such as normalization and missing values imputation. Finally, visualization tools are provided to explore the results, including heatmap, volcano plot and barplot representations. For scientists with limited experience in R, the package also contains wrapper functions that entail the complete analysis workflow and generate a report. Even easier to use are the interactive Shiny apps that are provided by the package.

License Artistic-2.0

Depends R (>= 3.5)

Encoding UTF-8

LazyData true

Imports ggplot2, dplyr, purrr, readr, tibble, tidyr,
    SummarizedExperiment (>= 1.11.5), MSnbase, limma, vsn, fdrtool,
    ggrepel, ComplexHeatmap, RColorBrewer, circlize, shiny,
    shinydashboard, DT, rmarkdown, assertthat, gridExtra, grid,
    stats, imputeLCMD, cluster

RoxygenNote 6.1.1

Suggests testthat, enrichR, knitr, BiocStyle

biocViews ImmunoOncology, Proteomics, MassSpectrometry,
    DifferentialExpression, DataRepresentation

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/DEP
Contents

- add_rejections .................................................. 3
- analyze_dep ..................................................... 4
- DEP .............................................................. 5
- DiUbi ............................................................. 7
- DiUbi_ExpDesign ................................................ 8
- filter_missval .................................................... 9
- filter_proteins .................................................. 10
- get_df_long ...................................................... 11
- get_df_wide ..................................................... 12
- get_prefix ....................................................... 13
- get_results ...................................................... 13
- get_suffix ....................................................... 14
- import_IsobarQuant ............................................. 15
- import_MaxQuant ............................................... 16
- impute ........................................................... 17
- LFQ ............................................................... 18
- make_se .......................................................... 19
- make_se_parse ................................................... 20
- make_unique .................................................... 21
- manual_impute .................................................. 22
- meanSdPlot ....................................................... 23
- normalize_vsn .................................................. 24
- plot_all .......................................................... 24
- plot_cond ........................................................ 25
- plot_cond_freq .................................................. 26
- plot_cond_overlap .............................................. 27
- plot_cor ........................................................ 28
- plot_coverage ................................................... 30
- plot_detect ...................................................... 30
- plot_dist ........................................................ 31
- plot_frequency .................................................. 32
- plot_gsea ........................................................ 33
- plot_heatmap ..................................................... 34
- plot_imputation ................................................ 36
- plot_missval ..................................................... 37
add_rejections

Mark significant proteins

Description

add_rejections marks significant proteins based on defined cutoffs.

Usage

add_rejections(diff, alpha = 0.05, lfc = 1)

Arguments

diff SummarizedExperiment, Proteomics dataset on which differential enrichment analysis has been performed (output from test_diff()).
alpha Numeric(1), Sets the threshold for the adjusted P value.
lfc Numeric(1), Sets the threshold for the log2 fold change.

Value

A SummarizedExperiment object annotated with logical columns indicating significant proteins.
Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)
```

---

**analyze_dep**

**Differential expression analysis**

**Description**

`analyze_dep` tests for differential expression of proteins based on protein-wise linear models and empirical Bayes statistics using `limma`.

**Usage**

```r
analyze_dep(se, type = c("all", "control", "manual"), control = NULL,
alpha = 0.05, lfc = 1, test = NULL, design_formula = formula(~0 +
condition))
```

**Arguments**

- **se** SummarizedExperiment, Proteomics data with unique names and identifiers annotated in 'name' and 'ID' columns. Additionally, the colData should contain sample annotation including 'label', 'condition' and 'replicate' columns. The appropriate columns and objects can be generated using `make_se` or `make_se_parse`.
- **type** "all", "control" or "manual", The type of contrasts that will be tested. This can be all possible pairwise comparisons ("all"), limited to the comparisons versus the control ("control"), or manually defined contrasts ("manual").
- **control** Character(1), The condition to which contrasts are generated (a control condition would be most appropriate).
- **alpha** Numeric(1), Sets the threshold for the adjusted P value.
- **lfc** Numeric(1), Sets the threshold for the log2 fold change.
test  Character, The contrasts that will be tested if type = "manual". These should be formatted as "SampleA_vs_SampleB" or c("SampleA_vs_SampleC", "SampleB_vs_SampleC").

design_formula  Formula, Used to create the design matrix.

Value

A SummarizedExperiment object containing FDR estimates of differential expression and logical columns indicating significant proteins.

Examples

```r
# Load datasets
data <- UbiLength
exp_design <- UbiLength_ExpDesign

# Import and process data
se <- import_MaxQuant(data, exp_design)
processed <- process(se)

# Differential protein expression analysis
dep <- analyze_dep(processed, "control", "Ctrl")
dep <- analyze_dep(processed, "control", "Ctrl",
    alpha = 0.01, lfc = log2(1.5))
dep <- analyze_dep(processed, "manual", test = c("Ubi6_vs_Ubi4"))
```

DEP

*DEP*: A package for Differential Enrichment analysis of Proteomics data.

Description

This package provides an integrated analysis workflow for robust and reproducible analysis of mass spectrometry proteomics data for differential protein expression or differential enrichment. It requires tabular input (e.g. txt files) as generated by quantitative analysis softwares of raw mass spectrometry data, such as MaxQuant or IsobarQuant. Functions are provided for data preparation, filtering, variance normalization and imputation of missing values, as well as statistical testing of differentially enriched / expressed proteins. It also includes tools to check intermediate steps in the workflow, such as normalization and missing values imputation. Finally, visualization tools are provided to explore the results, including heatmap, volcano plot and barplot representations. For scientists with limited experience in R, the package also entails wrapper functions that entail the complete analysis workflow and generate a report. Even easier to use are the interactive Shiny apps that are provided by the package.

Shiny apps

- `run_app`: Shiny apps for interactive analysis.
Workflow functions

- **LFQ**: Label-free quantification (LFQ) workflow wrapper.
- **TMT**: Tandem-mass-tags (TMT) workflow wrapper.
- **report**: Create a rmarkdown report wrapper.

Wrapper functions

- **import_MaxQuant**: Import data from MaxQuant into a SummarizedExperiment object.
- **import_IsobarQuant**: Import data from IsobarQuant into a SummarizedExperiment object.
- **process**: Perform filtering, normalization and imputation on protein data.
- **analyze_dep**: Differential protein expression analysis.
- **plot_all**: Visualize the results in different types of plots.

Main functions

- **make_unique**: Generate unique names.
- **make_se_parse**: Turn data.frame into SummarizedExperiment by parsing column names.
- **make_se**: Turn data.frame into SummarizedExperiment using an experimental design.
- **filter_proteins**: Filter proteins based on missing values.
- **normalize_vsn**: Normalize data using vsn.
- **impute**: Impute missing values.
- **test_diff**: Differential enrichment analysis.
- **add_rejections**: Mark significant proteins.
- **get_results**: Generate a results table.

Visualization functions

- **plot_single**: Barplot for a protein of interest.
- **plot_volcano**: Volcano plot for a specified contrast.
- **plot_heatmap**: Heatmap of all significant proteins.
- **plot_normalization**: Boxplots to inspect normalization.
- **plot_detect**: Density and CumSum plots of proteins with and without missing values.
- **plot_imputation**: Density plots to inspect imputation.
- **plot_missval**: Heatmap to inspect missing values.
- **plot_numbers**: Barplot of proteins identified.
- **plot_frequency**: Barplot of protein identification overlap between conditions.
- **plot_coverage**: Barplot of the protein coverage in conditions.
- **plot_pca**: PCA plot of top variable proteins.
- **plot_cor**: Plot correlation matrix.
- **plot_cor**: Plot Gower’s distance matrix.
• **plot_p_hist**: P value histogram.
• **plot_cond_freq**: Barplot of the number of significant conditions per protein.
• **plot_cond_overlap**: Barplot of the number of proteins for overlapping conditions.
• **plot_cond**: Barplot of the frequency of significant conditions per protein and the overlap in proteins between conditions.

**Gene Set Enrichment Analysis functions**
• **test_gsea**: Gene Set Enrichment Analysis using enrichR.
• **plot_gsea**: Barplot of enriched gene sets.

**Additional functions**
• **get_df_wide**: Generate a wide data.frame from a SummarizedExperiment.
• **get_df_long**: Generate a long data.frame from a SummarizedExperiment.
• **se2msn**: SummarizedExperiment object to MSnSet object conversion.
• **filter_missval**: Filter on missing values.
• **manual_impute**: Imputation by random draws from a manually defined distribution.
• **get_prefix**: Obtain the longest common prefix.
• **get_suffix**: Obtain the longest common suffix.

**Example data**
• **UbiLength**: Ubiquitin interactors of different linear ubiquitin lengths (UbIA-MS dataset) (Zhang, Smits, van Tilburg et al. Mol. Cell 2017).
• **UbiLength_ExpDesign**: Experimental design of the UbiLength dataset.
• **DiUbi**: Ubiquitin interactors for different diubiquitin-linkages (UbIA-MS dataset) (Zhang, Smits, van Tilburg et al. Mol. Cell 2017).
• **DiUbi_ExpDesign**: Experimental design of the DiUbi dataset.

---

**DiUbi**

*DiUbi - Ubiquitin interactors for different diubiquitin-linkages (UbIA-MS dataset)*

**Description**

The DiUbi dataset contains label free quantification (LFQ) and intensity-based absolute quantification (iBAQ) data for ubiquitin interactors of different diubiquitin-linkages, generated by Zhang et al 2017. The dataset contains the proteingroups output file from MaxQuant.

**Usage**

DiUbi
Format

A data.frame with 4071 observations and 102 variables:

- **Protein.IDs** Uniprot IDs
- **Majority.protein.IDs** Uniprot IDs of major protein(s) in the protein group
- **Protein.names** Full protein names
- **Gene.names** Gene name
- **Fasta.headers** Header as present in the Uniprot fasta file
- **Peptides** Number of peptides identified for this protein group
- **Razor...unique.peptides** Number of peptides used for the quantification of this protein group
- **Unique.peptides** Number of peptides identified which are unique for this protein group
- **Intensity columns (30)** Raw mass spectrometry intensity, A.U.
- **iBAQ columns (30)** iBAQ normalized mass spectrometry intensity, A.U.
- **LFQ.intensity columns (30)** LFQ normalized mass spectrometry intensity, A.U.
- **Only.identified.by.site** The protein is only identified by a modification site if marked (‘+’)
- **Reverse** The protein is identified in the decoy database if marked (‘+’)
- **Potential.contaminant** The protein is a known contaminant if marked (‘+’)
- **id** The protein group ID

Value

A data.frame.

Source


---

**DiUbi_ExpDesign**

Experimental design of the DiUbi dataset

Description

The DiUbi_ExpDesign object annotates 30 different samples of the DiUbi dataset in 10 conditions and 3 replicates.

Usage

DiUbi_ExpDesign
**filter_missval**

**Format**

A data.frame with 30 observations and 3 variables:

- **label**  Label names
- **condition**  Experimental conditions
- **replicate**  Replicate number

**Value**

A data.frame.

**Source**


---

**filter_missval**  Filter on missing values

**Description**

filter_missval filters a proteomics dataset based on missing values. The dataset is filtered for proteins that have a maximum of 'thr' missing values in at least one condition.

**Usage**

```r
filter_missval(se, thr = 0)
```

**Arguments**

- **se**  SummarizedExperiment, Proteomics data (output from `make_se()` or `make_se_parse()`).
- **thr**  Integer(1), Sets the threshold for the allowed number of missing values in at least one condition.

**Value**

A filtered SummarizedExperiment object.

**Examples**

# Load example
data <- UbiLength
data <- data[data$Reverse != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
filter_proteins

Filter proteins based on missing values

Description

filter_proteins filters a proteomic dataset based on missing values. Different types of filtering can be applied, which range from only keeping proteins without missing values to keeping proteins with a certain percent valid values in all samples or keeping proteins that are complete in at least one condition.

Usage

filter_proteins(se, type = c("complete", "condition", "fraction"),
thr = NULL, min = NULL)

Arguments

se
SummarizedExperiment, Proteomics data (output from make_se() or make_se_parse()).

type
"complete", "condition" or "fraction", Sets the type of filtering applied. "complete" will only keep proteins with valid values in all samples. "condition" will keep proteins that have a maximum of 'thr' missing values in at least one condition. "fraction" will keep proteins that have a certain fraction of valid values in all samples.

thr
Integer(1), Sets the threshold for the allowed number of missing values in at least one condition if type = "condition".

min
Numeric(1), Sets the threshold for the minimum fraction of valid values allowed for any protein if type = "fraction".

Value

A filtered SummarizedExperiment object.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

data <- grep("LFQ.", colnames(data_unique))
experiment <- UbiLength_ExpDesign

# Filter
stringent_filter <- filter_missval(se, thr = 0)
less_stringent_filter <- filter_missval(se, thr = 1)
get_df_long

se <- make_se(data_unique, columns, exp_design)

# Filter
stringent_filter <- filter_proteins(se, type = "complete")
less_stringent_filter <- filter_proteins(se, type = "condition", thr = 0)

get_df_long

Generate a long data.frame from a SummarizedExperiment

Description

get_df_long generate a wide data.frame from a SummarizedExperiment.

Usage

get_df_long(se)

Arguments

se SummarizedExperiment, Proteomics data (output from make_se() or make_se_parse()).

Value

A data.frame object containing all data in a wide format, where each row represents a single measurement.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exposure <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Get a long data.frame
long <- get_df_long(dep)
colnames(long)
get_df_wide

Generate a wide data.frame from a SummarizedExperiment

Description

get_df_wide generate a wide data.frame from a SummarizedExperiment.

Usage

get_df_wide(se)

Arguments

se

SummarizedExperiment, Proteomics data (output from make_se() or make_se_parse()).

Value

A data.frame object containing all data in a wide format, where each row represents a protein.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+
& data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Get a wide data.frame
wide <- get_df_wide(dep)
colnames(wide)
get_prefix

Obtain the longest common prefix

Description
get_prefix returns the longest common prefix of the supplied words.

Usage
get_prefix(words)

Arguments
words Character vector, A list of words.

Value
A character vector containing the prefix.

Examples
# Load example
data <- UbiLength
columns <- grep("LFQ.", colnames(data))

# Get prefix
names <- colnames(data[, columns])
get_prefix(names)

generate a results table

Description
generate a results table from a proteomics dataset on which differential enrichment analysis was performed.

Usage
generate_results(dep)

Arguments
dep SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from test_diff() and add_rejections()).
Value

A data.frame object containing all results variables from the performed analysis.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Get results
results <- get_results(dep)
colnames(results)

significant_proteins <- results[results$significant,
nrow(significant_proteins)
head(significant_proteins)

get_suffix

Obtain the longest common suffix

Description

get_suffix returns the longest common suffix of the supplied words.

Usage

get_suffix(words)

Arguments

words Character vector, A list of words.

Value

A character vector containing the suffix
import_IsobarQuant

Examples

# Get suffix
names <- c("xyz_rep", "abc_rep")
get_suffix(names)

import_IsobarQuant Import from IsobarQuant

Description

import_IsobarQuant imports a protein table from IsobarQuant and converts it into a SummarizedExperiment object.

Usage

import_IsobarQuant(proteins, expdesign, intensities = "signal_sum",
                   names = "gene_name", ids = "protein_id", delim = "[|]")

Arguments

proteins Data.frame, Protein table for which unique names will be created.
expdesign Data.frame, Experimental design with 'label', 'condition' and 'replicate' information. See UbiLength_ExpDesign for an example experimental design.
intensities Character(1), Prefix of the columns containing sample intensities.
names Character(1), Name of the column containing feature names.
ids Character(1), Name of the column containing feature IDs.
delim Character(1), Sets the delimiter separating the feature names within on protein group.

Value

A SummarizedExperiment object with log2-transformed values and "name" and "ID" columns containing unique names and identifiers.

Examples

## Not run:
# Load data
isobarquant_table <- read.csv("testfile.txt", header = TRUE,
                               stringsAsFactors = FALSE, sep = "\t")
exp_design <- read.csv("test_experimental_design.txt", header = TRUE,
                        stringsAsFactors = FALSE, sep = "\t")
# Import data
se <- import_IsobarQuant(isobarquant_table, exp_design)

## End(Not run)
import_MaxQuant

Import from MaxQuant

Description

import_MaxQuant imports a protein table from MaxQuant and converts it into a SummarizedExperiment object.

Usage

import_MaxQuant(proteins, expdesign, filter = c("Reverse", "Potential.contaminant"), intensities = "LFQ", names = "Gene.names", ids = "Protein.IDs", delim = ";")

Arguments

- **proteins**: Data.frame, Protein table originating from MaxQuant.
- **expdesign**: Data.frame, Experimental design with 'label', 'condition' and 'replicate' information. See UbiLength_ExpDesign for an example experimental design.
- **filter**: Character, Name of the column(s) containing features to be filtered on.
- **intensities**: Character(1), Prefix of the columns containing sample intensities.
- **names**: Character(1), Name of the column containing feature names.
- **ids**: Character(1), Name of the column containing feature IDs.
- **delim**: Character(1), Sets the delimiter separating the feature names within on protein group.

Value

A SummarizedExperiment object with log2-transformed values and "name" and "ID" columns containing unique names and identifiers.

Examples

```r
# Load example data and experimental design
data <- UbiLength
exp_design <- UbiLength_ExpDesign

# Import data
se <- import_MaxQuant(data, exp_design)
```
impute  

**Description**

impute imputes missing values in a proteomics dataset.

**Usage**

```r
impute(se, fun = c("bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "man", "min", "zero", "mixed", "nbavg"), ...)
```

**Arguments**

- **se**: SummarizedExperiment, Proteomics data (output from `make_se()` or `make_se_parse()`). It is advised to first remove proteins with too many missing values using `filter_missval()` and normalize the data using `normalize_vsn()`.
- **fun**: "bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "man", "min", "zero", "mixed" or "nbavg", Function used for data imputation based on `manual_impute` and `impute`.
- **...**: Additional arguments for imputation functions as depicted in `manual_impute` and `impute`.

**Value**

An imputed SummarizedExperiment object.

**Examples**

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";;")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and normalize
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)

# Impute missing values using different functions
imputed_MinProb <- impute(norm, fun = "MinProb", q = 0.05)
imputed_QRILC <- impute(norm, fun = "QRILC")
imputed_knn <- impute(norm, fun = "knn", k = 10, rowmax = 0.9)
```
imputed_MLE <- impute(norm, fun = "MLE")
imputed_manual <- impute(norm, fun = "man", shift = 1.8, scale = 0.3)

LFQ workflow

Description

LFQ is a wrapper function running the entire differential enrichment/expression analysis workflow for label free quantification (LFQ)-based proteomics data. The protein table from MaxQuant is used as direct input.

Usage

LFQ(proteins, expdesign, fun = c("man", "b pca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "min", "zero", "mixed", "nbavg"), type = c("all", "control", "manual"), control = NULL, test = NULL, filter = c("Reverse", "Potential.contaminant"), name = "Gene.names", ids = "Protein.IDs", alpha = 0.05, lfc = 1)

Arguments

- proteins: Data.frame, The data object.
- expdesign: Data.frame, The experimental design object.
- type: 'all', 'control' or 'manual', The type of contrasts that will be generated.
- control: Character(1), The sample name to which the contrasts are generated (the control sample would be most appropriate).
- test: Character, The contrasts that will be tested if type = "manual". These should be formatted as "SampleA_vs_SampleB" or c("SampleA_vs_SampleC", "SampleB_vs_SampleC").
- filter: Character, Name(s) of the column(s) to be filtered on.
- name: Character(1), Name of the column representing gene names.
- ids: 'Character(1), Name of the column representing protein IDs.
- alpha: Numeric(1), sets the false discovery rate threshold.
- lfc: Numeric(1), sets the log fold change threshold.
**make_se**

### Value

A list of 9 objects:
- `data`: data.frame containing the original data
- `se`: SummarizedExperiment object containing the original data
- `filt`: SummarizedExperiment object containing the filtered data
- `norm`: SummarizedExperiment object containing the normalized data
- `imputed`: SummarizedExperiment object containing the imputed data
- `diff`: SummarizedExperiment object containing FDR estimates of differential expression
- `dep`: SummarizedExperiment object annotated with logical columns indicating significant proteins
- `results`: data.frame containing containing all results variables from the performed analysis
- `param`: data.frame containing the test parameters

### Examples

```r
data <- UbiLength
expdesign <- UbiLength_ExpDesign
results <- LFQ(data, expdesign, 'MinProb', 'control', 'Ctrl')
```

---

### Description

`make_se` creates a SummarizedExperiment object based on two data.frames: the protein table and experimental design.

### Usage

`make_se(proteins_unique, columns, expdesign)`

### Arguments

- `proteins_unique`: Data.frame, Protein table with unique names annotated in the 'name' column (output from `make_unique()`).
- `columns`: Integer vector, Column numbers indicating the columns containing the assay data.
- `expdesign`: Data.frame, Experimental design with 'label', 'condition' and 'replicate' information. See `UbiLength_ExpDesign` for an example experimental design.
**Value**

A SummarizedExperiment object with log2-transformed values.

**Examples**

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+"] & data$Potential.contaminant != "+.

data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)
```

---

### Description

`make_se_parse` creates a SummarizedExperiment object based on a single data.frame.

### Usage

```r
make_se_parse(proteins_unique, columns, mode = c("char", "delim"),
chars = 1, sep = ";")
```

### Arguments

- `proteins_unique`: Data.frame, Protein table with unique names annotated in the 'name' column (output from `make_unique()`).
- `columns`: Integer vector, Column numbers indicating the columns containing the assay data.
- `mode`: "char" or "delim", The mode of parsing the column headers. "char" will parse the last number of characters as replicate number and requires the 'chars' parameter. "delim" will parse on the separator and requires the 'sep' parameter.
- `chars`: Numeric(1), The number of characters to take at the end of the column headers as replicate number (only for mode == "char").
- `sep`: Character(1), The separator used to parse the column header (only for mode == "delim").

### Value

A SummarizedExperiment object with log2-transformed values.
Examples

```r
# Load example
data <- Ubilength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
se <- make_se_parse(data_unique, columns, mode = "char", chars = 1)
se <- make_se_parse(data_unique, columns, mode = "delim", sep = "_")
```

make_unique

Make unique names

Description

make_unique generates unique identifiers for a proteomics dataset based on "name" and "id" columns.

Usage

```r
make_unique(proteins, names, ids, delim = ";")
```

Arguments

- **proteins**: Data.frame, Protein table for which unique names will be created.
- **names**: Character(1), Name of the column containing feature names.
- **ids**: Character(1), Name of the column containing feature IDs.
- **delim**: Character(1), Sets the delimiter separating the feature names within one protein group.

Value

A data.frame with the additional variables "name" and "ID" containing unique names and identifiers, respectively.

Examples

```r
# Load example
data <- Ubilength

# Check colnames and pick the appropriate columns
colnames(data)
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")
```
manual_impute

Imputation by random draws from a manually defined distribution

Description

manual_impute imputes missing values in a proteomics dataset by random draws from a manually defined distribution.

Usage

manual_impute(se, scale = 0.3, shift = 1.8)

Arguments

se SummarizedExperiment, Proteomics data (output from make_se() or make_se_parse()). It is advised to first remove proteins with too many missing values using filter_missval() and normalize the data using normalize_vsn().

scale Numeric(1), Sets the width of the distribution relative to the standard deviation of the original distribution.

shift Numeric(1), Sets the left-shift of the distribution (in standard deviations) from the median of the original distribution.

Value

An imputed SummarizedExperiment object.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+"] & data$Potential.contaminant != "+

data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ;;

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and normalize
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)

# Impute missing values manually
imputed_manual <- impute(norm, fun = "man", shift = 1.8, scale = 0.3)
meanSdPlot

Plot row standard deviations versus row means

Description

meanSdPlot generates a hexagonal heatmap of the row standard deviations versus row means from SummarizedExperiment objects. See meanSdPlot.

Usage

meanSdPlot(x, ranks = TRUE, xlab = ifelse(ranks, "rank(mean)", "mean"), ylab = "sd", pch, plot = TRUE, bins = 50, ...)

Arguments

- **x**: SummarizedExperiment, Data object.
- **ranks**: Logical, Whether or not to plot the row means on the rank scale.
- **xlab**: Character, x-axis label.
- **ylab**: Character, y-axis label.
- **pch**: Ignored - exists for backward compatibility.
- **plot**: Logical, Whether or not to produce the plot.
- **bins**: Numeric vector, Data object before normalization.
- **...**: Other arguments, Passed to stat_binhex.

Value

A scatter plot of row standard deviations versus row means (generated by stat_binhex)

Examples

```r
# Load example
data <- UbiLength
data <- data[, data$Reverse != "+", & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and normalize
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)

# Plot meanSdPlot
meanSdPlot(norm)
```
normalize_vsn  
*Normalization using vsn*

**Description**

normalize_vsn performs variance stabilizing transformation using the `vsn-package`.

**Usage**

```r
normalize_vsn(se)
```

**Arguments**

- `se`  
  SummarizedExperiment, Proteomics data (output from `make_se()` or `make_se_parse()`). It is advised to first remove proteins with too many missing values using `filter_missval()`.

**Value**

A normalized SummarizedExperiment object.

**Examples**

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+"] & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and normalize
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
```

---

plot_all  
*Visualize the results in different types of plots*

**Description**

plot_all visualizes the results of the differential protein expression analysis in different types of plots. These are (1) volcano plots, (2) heatmaps, (3) single protein plots, (4) frequency plots and/or (5) comparison plots.
plot_cond

Usage

plot_all(dep, plots = c("volcano", "heatmap", "single", "freq", "comparison"))

Arguments

dep SummarizedExperiment, Data object which has been generated by `analyze_dep` or the combination of `test_diff` and `add_rejections`.
plots "volcano", "heatmap", "single", "freq" and/or "comparison",

Value

Pdfs containing the desired plots.

Examples

# Load datasets
data <- UbiLength
exp_design <- UbiLength_ExpDesign

# Import and process data
se <- import_MaxQuant(data, exp_design)
processed <- process(se)

# Differential protein expression analysis
dep <- analyze_dep(processed, "control", "Ctrl")

## Not run:
# Plot all plots
plot_all(dep)

## End(Not run)

plot_cond

Plot frequency of significant conditions per protein and the overlap in proteins between conditions

Description

plot_cond generates a histogram of the number of proteins per condition and stacks for overlapping conditions.

Usage

plot_cond(dep, plot = TRUE)
Arguments

**dep** SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).

**plot** Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the data which the barplot is based on are returned.

Value

A histogram (generated by `ggplot`)

Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";"

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot histogram with overlaps
plot_cond(dep)
```

---

**plot_cond_freq**  
*Plot frequency of significant conditions per protein*

Description

`plot_cond_freq` generates a histogram of the number of significant conditions per protein.

Usage

```r
plot_cond_freq(dep, plot = TRUE)
```
Arguments

- **dep**: SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- **plot**: Logical(1), If TRUE (default) the histogram is produced. Otherwise (if FALSE), the data which the histogram is based on are returned.

Value

A histogram (generated by `ggplot`)

Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot frequency of significant conditions
plot_cond_freq(dep)
```

Description

`plot_cond_overlap` generates a histogram of the number of proteins per condition or overlapping conditions.

Usage

`plot_cond_overlap(dep, plot = TRUE)`
Arguments

- **dep**: SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- **plot**: Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the data which the barplot is based on are returned.

Value

A histogram (generated by `ggplot`)

Examples

```r
# Load example data
data <- UbiLength
data <- data[!data$Reverse == "+"]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
expp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot condition overlap
plot_cond_overlap(dep)
```

Description

`plot_cor` generates a Pearson correlation matrix.

Usage

```r
plot_cor(dep, significant = TRUE, lower = -1, upper = 1,
apal = "PRGn", pal_rev = FALSE, indicate = NULL, font_size = 12,
plot = TRUE, ...)
```
Arguments

- `dep` SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- `significant` Logical(1), Whether or not to filter for significant proteins.
- `lower` Integer(1), Sets the lower limit of the color scale.
- `upper` Integer(1), Sets the upper limit of the color scale.
- `pal` Character(1), Sets the color panel (from `RColorBrewer`).
- `pal_rev` Logical(1), Whether or not to invert the color palette.
- `indicate` Character, Sets additional annotation on the top of the heatmap based on columns from the experimental design (colData).
- `font_size` Integer(1), Sets the size of the labels.
- `plot` Logical(1), If `TRUE` (default) the correlation matrix plot is produced. Otherwise (if `FALSE`), the data which the correlation matrix plot is based on are returned.
  
  Additional arguments for Heatmap function as depicted in `Heatmap`

Value

A heatmap plot (generated by `Heatmap`)

Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot correlation matrix
plot_cor(dep)
```
plot_coverage  
Plot protein coverage

Description

plot_coverage generates a barplot of the protein coverage in all samples.

Usage

plot_coverage(se, plot = TRUE)

Arguments

- se: SummarizedExperiment, Data object for which to plot observation frequency.
- plot: Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the data which the barplot is based on are returned.

Value

Barplot of protein coverage in samples (generated by ggplot)

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+"] & data$Potential.contaminant != "+

data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
experiment <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, experiment)

# Filter and plot coverage
filt <- filter_missval(se, thr = 0)
plot_coverage(filt)

plot_detect  
Visualize intensities of proteins with missing values

Description

plot_detect generates density and CumSum plots of protein intensities with and without missing values.
**plot_dist**

*Plot Gower’s distance matrix*

**Description**

plot_dist generates a distance matrix heatmap using the Gower’s distance.

**Usage**

```r
plot_dist(dep, significant = TRUE, pal = "YlOrRd", pal_rev = TRUE, 
indicate = NULL, font_size = 12, plot = TRUE, ...)
```

**Arguments**

- `dep` SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- `significant` Logical(1), Whether or not to filter for significant proteins.
- `pal` Character(1), Sets the color panel (from `RColorBrewer`).

**Usage**

```r
plot_detect(se)
```

**Arguments**

- `se` SummarizedExperiment, Data object with missing values.

**Value**

Density and CumSum plots of intensities of proteins with and without missing values (generated by `ggplot`).

**Examples**

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter
filt <- filter_missval(se, thr = 0)

# Plot intensities of proteins with missing values
plot_detect(filt)
```
plot_frequency

pal_rev: Logical(1), Whether or not to invert the color palette.

indicate: Character, Sets additional annotation on the top of the heatmap based on columns from the experimental design (colData).

font_size: Integer(1), Sets the size of the labels.

plot: Logical(1), If TRUE (default) the distance matrix plot is produced. Otherwise (if FALSE), the data which the distance matrix plot is based on are returned.

... Additional arguments for Heatmap function as depicted in Heatmap.

Value

A heatmap plot (generated by Heatmap)

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot correlation matrix
plot_dist(dep)

plot_frequency

Plot protein overlap between samples

Description

plot_frequency generates a barplot of the protein overlap between samples.

Usage

plot_frequency(se, plot = TRUE)
plot_gsea

Arguments

se SummarizedExperiment, Data object for which to plot observation frequency.
plot Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the
data which the barplot is based on are returned.

Value

Barplot of overlap of protein identifications between samples (generated by ggplot)

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+"] & data$Potential.contaminant != "+"]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and plot frequency
filt <- filter_missval(se, thr = 0)
plot_frequency(filt)

plot_gsea

Plot enriched Gene Sets

Description

plot_gsea plots enriched gene sets from Gene Set Enrichment Analysis.

Usage

plot_gsea(gsea_results, number = 10, alpha = 0.05, contrasts = NULL,
databases = NULL, nrow = 1, term_size = 8)

Arguments

gsea_results Data.frame, Gene Set Enrichment Analysis results object. (output from test_gsea()).
number Numeric(1), Sets the number of enriched terms per contrast to be plotted.
alpha Numeric(1), Sets the threshold for the adjusted P value.
contrasts Character, Specifies the contrast(s) to plot. If ‘NULL’ all contrasts will be plotted.
databases Character, Specifies the database(s) to plot. If ‘NULL’ all databases will be plotted.
nrow Numeric(1), Sets the number of rows for the plot.
term_size Numeric(1), Sets the text size of the terms.
Value

A barplot of the enriched terms (generated by \texttt{ggplot}).

Examples

\begin{verbatim}
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", names(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.99)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

## Not run:

# Test enrichments
gsea_results <- test_gsea(dep)
plot_gsea(gsea_results)

## End(Not run)
\end{verbatim}

---

\textbf{plot\_heatmap} \textit{Plot a heatmap}

Description

\texttt{plot\_heatmap} generates a heatmap of all significant proteins.

Usage

\begin{verbatim}
plot\_heatmap\(\texttt{(dep, type = \texttt{c("contrast", "centered")}, kmeans = \texttt{FALSE},
k = \texttt{6}, col\_limit = \texttt{6}, indicate = \texttt{NULL},
clustering\_distance = \texttt{c("euclidean", "maximum", "manhattan",
"canberra", "binary", "minkowski", "pearson", "spearman", "kendall",
gower"), row\_font\_size = \texttt{6}, col\_font\_size = \texttt{10}, plot = \texttt{TRUE, ...})}
\end{verbatim}
plot_heatmap

Arguments

- **dep**: SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- **type**: 'contrast' or 'centered', The type of data scaling used for plotting. Either the fold change ('contrast') or the centered log2-intensity ('centered').
- **kmeans**: Logical(1), Whether or not to perform k-means clustering.
- **k**: Integer(1), Sets the number of k-means clusters.
- **col_limit**: Integer(1), Sets the outer limits of the color scale.
- **indicate**: Character, Sets additional annotation on the top of the heatmap based on columns from the experimental design (colData). Only applicable to type = 'centered'.
- **clustering_distance**: "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "pearson", "spearman", "kendall" or "gower", Function used to calculate clustering distance (for proteins and samples). Based on `Heatmap` and `daisy`.
- **row_font_size**: Integer(1), Sets the size of row labels.
- **col_font_size**: Integer(1), Sets the size of column labels.
- **plot**: Logical(1), If TRUE (default) the heatmap is produced. Otherwise (if FALSE), the data which the heatmap is based on are returned.
- **...**: Additional arguments for Heatmap function as depicted in `Heatmap`

Value

A heatmap (generated by `Heatmap`)

Examples

```
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot heatmap
plot_heatmap(dep)
```
plot_heatmap(dep, 'centered', kmeans = TRUE, k = 6, row_font_size = 3)
plot_heatmap(dep, 'contrast', col_limit = 10, row_font_size = 3)

---

**plot_imputation**

**Visualize imputation**

**Description**

plot_imputation generates density plots of all conditions for input objects, e.g. before and after imputation.

**Usage**

plot_imputation(se, ...)

**Arguments**

- **se**
  SummarizedExperiment, Data object, e.g. before imputation (output from `normalize_vsn()`).
- **...**
  Other SummarizedExperiment object(s), E.g. data object after imputation (output from `impute()`).

**Value**

Density plots of all conditions of all conditions for input objects, e.g. before and after imputation (generated by `ggplot`).

**Examples**

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Plot imputation
plot_imputation(filt, norm, imputed)
```
plot_missval

Plot a heatmap of proteins with missing values

Description
plot_missval generates a heatmap of proteins with missing values to discover whether values are missing by random or not.

Usage
plot_missval(se)

Arguments
se SummarizedExperiment, Data object with missing values.

Value
A heatmap indicating whether values are missing (0) or not (1) (generated by Heatmap).

Examples
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)

# Plot missing values heatmap
plot_missval(filt)

plot_normalization

Visualize normalization

Description
plot_normalization generates boxplots of all conditions for input objects, e.g. before and after normalization.
Usage

plot_normalization(se, ...)

Arguments

se SummarizedExperiment, Data object, e.g. before normalization (output from make_se() or make_se_parse()).
...
Additional SummarizedExperiment object(s), E.g. data object after normalization (output from normalize_vsn).

Value

Boxplots of all conditions for input objects, e.g. before and after normalization (generated by ggplot). Adding components and other plot adjustments can be easily done using the ggplot2 syntax (i.e. using '+')

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and normalize
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)

# Plot normalization
plot_normalization(se, filt, norm)

Description

plot_numbers generates a barplot of the number of identified proteins per sample.

Usage

plot_numbers(se, plot = TRUE)
plot_pca

Arguments

se SummarizedExperiment, Data object for which to plot protein numbers (output from make_se() or make_se_parse()).
plot Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the data which the barplot is based on are returned.

Value

Barplot of the number of identified proteins per sample (generated by ggplot)

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+"]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter and plot numbers
filt <- filter_missval(se, thr = 0)
plot_numbers(filt)

plot_pca

Plot PCA

Description

plot_pca generates a PCA plot using the top variable proteins.

Usage

plot_pca(dep, x = 1, y = 2, indicate = c("condition", "replicate"),
label = FALSE, n = 500, point_size = 4, label_size = 3,
plot = TRUE)

Arguments

dep SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from test_diff() and add_rejections()).
x Integer(1), Sets the principle component to plot on the x-axis.
y Integer(1), Sets the principle component to plot on the y-axis.
indicate Character, Sets the color, shape and facet_wrap of the plot based on columns from the experimental design (colData).
plot_p_hist

Description

plot_p_hist generates a p value histogram.

Usage

plot_p_hist(dep, adjusted = FALSE, wrap = FALSE)
plot_single

Arguments

- **dep**: SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- **adjusted**: Logical(1), Whether or not to use adjusted p values.
- **wrap**: Logical(1), Whether or not to display different histograms for the different contrasts.

Value

A histogram (generated by `ggplot`).

Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+", & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot p value histogram
plot_p_hist(dep)
plot_p_hist(dep, wrap = TRUE)
```

**plot_single**  
*Plot values for a protein of interest*

Description

`plot_single` generates a barplot of a protein of interest.

Usage

```r
plot_single(dep, proteins, type = c("contrast", "centered"), plot = TRUE)
```
Arguments

- **dep**: SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from `test_diff()` and `add_rejections()`).
- **proteins**: Character, The name(s) of the protein(s) to plot.
- **type**: 'contrast' or 'centered', The type of data scaling used for plotting. Either the fold change ('contrast') or the centered log2-intensity ('centered').
- **plot**: Logical(1), If TRUE (default) the barplot is produced. Otherwise (if FALSE), the summaries which the barplot is based on are returned.

Value

A barplot (generated by `ggplot`).

Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+", data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot single proteins
plot_single(dep, 'USP15')
plot_single(dep, 'USP15', 'centered')
plot_single(dep, c('USP15', 'CUL1'))
plot_single(dep, c('USP15', 'CUL1'), plot = FALSE)
```

---

**plot_volcano**  
Volcano plot

Description

`plot_volcano` generates a volcano plot for a specified contrast.
plot_volcano

Usage

plot_volcano(dep, contrast, label_size = 3, add_names = TRUE, adjusted = FALSE, plot = TRUE)

Arguments

dep SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from test_diff() and add_rejections()).
contrast Character(1), Specifies the contrast to plot.
label_size Integer(1), Sets the size of name labels.
add_names Logical(1), Whether or not to plot names.
adjusted Logical(1), Whether or not to use adjusted p values.
plot Logical(1), If TRUE (default) the volcano plot is produced. Otherwise (if FALSE), the data which the volcano plot is based on are returned.

Value

A volcano plot (generated by ggplot)

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != + & data$Potential.contaminant != +,]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

# Plot volcano
plot_volcano(dep, 'Ubi6_vs_Ctrl', label_size = 5, add_names = TRUE)
plot_volcano(dep, 'Ubi6_vs_Ctrl', label_size = 5, add_names = TRUE, adjusted = TRUE)
plot_volcano(dep, 'Ubi6_vs_Ctrl', add_names = FALSE)
plot_volcano(dep, 'Ubi4_vs_Ctrl', label_size = 5, add_names = TRUE)
process Proteomics data processing

Description

process performs data processing on a SummarizedExperiment object. It (1) filters a proteomics dataset based on missing values, (2) applies variance stabilizing normalization and (3) imputes eventual remaining missing values.

Usage

process(se, thr = 0, fun = c("man", "bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "min", "zero", "mixed", "nbavg"), ...)

Arguments

se SummarizedExperiment, Proteomics data with unique names and identifiers annotated in ‘name’ and ‘ID’ columns. The appropriate columns and objects can be generated using the wrapper import functions import_MaxQuant and import_IsobarQuant or the generic functions make_se and make_se_parse.

thr Integer(1), Sets the threshold for the allowed number of missing values per condition.


... Additional arguments for imputation functions as depicted in manual_impute and impute.

Value

A filtered, normalized and imputed SummarizedExperiment object.

Examples

# Load datasets
data <- UbiLength
design <- UbiLength_ExpDesign

# Import data
se <- import_MaxQuant(data, design)

# Process data
processed <- process(se)
**report**  
*Generate a markdown report*

**Description**

`report` generates a report of the analysis performed by TMT and LFQ wrapper functions. Additionally, the results table is saved as a tab-delimited file.

**Usage**

```r
report(results)
```

**Arguments**

- `results` List of SummarizedExperiment objects obtained from the LFQ or TMT wrapper functions.

**Value**

A *rmarkdown* report is generated and saved. Additionally, the results table is saved as a tab-delimited txt file.

**Examples**

```r
## Not run:
data <- UbiLength
expsdesign <- UbiLength_ExpDesign
results <- LFQ(data, expdesign, 'MinProb', 'control', 'Ctrl')
report(results)

## End(Not run)
```

---

**run_app**  
*DEP shiny apps*

**Description**

`run_app` launches an interactive shiny app for interactive differential enrichment/expression analysis of proteomics data.

**Usage**

```r
run_app(app)
```
Arguments

app 'LFQ' or 'TMT', The name of the app.

Value

Launches a browser with the shiny app

Examples

## Not run:
# Run the app
run_app('LFQ')
run_app('TMT')

## End(Not run)

---

se2msn

**Deprecated Function to coerce SummarizedExperiment to MSnSet object**

Description

Use `as` instead.

Usage

se2msn(se)

Arguments

se SummarizedExperiment, Object which will be turned into a MSnSet object.

Value

A MSnSet object.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]

data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)
test_diff

# Convert to MSnSet
data_msn <- as(se, "MSnSet")
# Convert back to SE
se_back <- as(data_msn, "SummarizedExperiment")

diff <- test_diff(se, type = c("control", "all", "manual"), control = NULL,
test = NULL, design_formula = formula(~0 + condition))

Description

test_diff performs a differential enrichment test based on protein-wise linear models and empirical Bayes statistics using limma. False Discovery Rates are estimated using fdrtool.

Usage

test_diff(se, type = c("control", "all", "manual"), control = NULL,
test = NULL, design_formula = formula(~0 + condition))

Arguments

se  SummarizedExperiment, Proteomics data (output from make_se() or make_se_parse()). It is advised to first remove proteins with too many missing values using filter_missval(), normalize the data using normalize_vsn() and impute remaining missing values using impute().
type  "control", "all" or "manual", The type of contrasts that will be tested. This can be all possible pairwise comparisons ("all"), limited to the comparisons versus the control ("control"), or manually defined contrasts ("manual").
control  Character(1), The condition to which contrasts are generated if type = "control" (a control condition would be most appropriate).
test  Character, The contrasts that will be tested if type = "manual". These should be formatted as "SampleA_vs_SampleB" or c("SampleA_vs_SampleC", "SampleB_vs_SampleC").
design_formula  Formula, Used to create the design matrix.

Value

A SummarizedExperiment object containing fdr estimates of differential expression.

Examples

# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene.names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
diff <- test_diff(imputed, "manual",
    test = c("Ubi4_vs_Ctrl", "Ubi6_vs_Ctrl"))

# Test for differentially expressed proteins with a custom design formula
diff <- test_diff(imputed, "control", "Ctrl",
    design_formula = formula(~ 0 + condition + replicate))

---

**test_gsea**

*Gene Set Enrichment Analysis*

**Description**

test_gsea tests for enriched gene sets in the differentially enriched proteins. This can be done independently for the different contrasts.

**Usage**

test_gsea(dep, databases = c("GO_Molecular_Function_2017b",
"GO_Cellular_Component_2017b", "GO_Biological_Process_2017b"),
contrasts = TRUE)

**Arguments**

dep SummarizedExperiment, Data object for which differentially enriched proteins are annotated (output from test_diff() and add_rejections()).
databases Character, Databases to search for gene set enrichment. See http://amp.pharm.mssm.edu/Enrichr/ for available databases.
contrasts Logical(1), Whether or not to perform the gene set enrichment analysis independently for the different contrasts.

**Value**

A data.frame with enrichment terms (generated by enrichr)
Examples

```r
# Load example
data <- UbiLength
data <- data[data$Reverse != "+" & data$Potential.contaminant != "+",]
data_unique <- make_unique(data, "Gene_names", "Protein.IDs", delim = ";")

# Make SummarizedExperiment
columns <- grep("LFQ.", colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

# Filter, normalize and impute missing values
filt <- filter_missval(se, thr = 0)
norm <- normalize_vsn(filt)
imputed <- impute(norm, fun = "MinProb", q = 0.01)

# Test for differentially expressed proteins
diff <- test_diff(imputed, "control", "Ctrl")
dep <- add_rejections(diff, alpha = 0.05, lfc = 1)

## Not run:
# Test enrichments
gsea_results_per_contrast <- test_gsea(dep)
gsea_results <- test_gsea(dep, contrasts = FALSE)
gsea_kegg <- test_gsea(dep, databases = "KEGG_2016")

## End(Not run)
```

theme_DEP1

---

### Description

theme_DEP1 is the default ggplot theme used for plotting in DEP with horizontal x-axis labels.

### Usage

```r
theme_DEP1()
```

### Value

ggplot theme
Examples

```
data <- UbiLength
data <- data[data$Reverse != '+' & data$Potential.contaminant != '+',]
data_unique <- make_unique(data, 'Gene.names', 'Protein.IDs', delim = ';')

columns <- grep('LFQ.', colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

filt <- filter_missval(se, thr = 0)
plot_frequency(filt) # uses theme_DEP1() style
```

---

**theme_DEP2**  
*DEP ggplot theme 2*

Description

`theme_DEP2` is the ggplot theme used for plotting in DEP with vertical x-axis labels.

Usage

```
theme_DEP2()
```

Value

*ggplot theme*

Examples

```
data <- UbiLength
data <- data[data$Reverse != '+' & data$Potential.contaminant != '+',]
data_unique <- make_unique(data, 'Gene.names', 'Protein.IDs', delim = ';')

columns <- grep('LFQ.', colnames(data_unique))
exp_design <- UbiLength_ExpDesign
se <- make_se(data_unique, columns, exp_design)

filt <- filter_missval(se, thr = 0)
plot_numbers(filt) # uses theme_DEP2() style
```
TMT workflow

Description

TMT is a wrapper function running the entire differential enrichment/expression analysis workflow for TMT-based proteomics data. The protein table from IsobarQuant is used as direct input.

Usage

```r
TMT(proteins, expdesign, fun = c("man", "bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "min", "zero", "mixed", "nbavg"), type = c("all", "control", "manual"), control = NULL, test = NULL, name = "gene_name", ids = "protein_id", alpha = 0.05, lfc = 1)
```

Arguments

- `proteins` Data.frame, The data object.
- `expdesign` Data.frame, The experimental design object.
- `type` 'all', 'control' or 'manual', The type of contrasts that will be generated.
- `control` Character(1), The sample name to which the contrasts are generated (the control sample would be most appropriate).
- `test` Character, The contrasts that will be tested if type = "manual". These should be formatted as "SampleA_vs_SampleB" or c("SampleA_vs_SampleC", "SampleB_vs_SampleC").
- `name` Character(1), Name of the column representing gene names.
- `ids` Character(1), Name of the column representing protein IDs.
- `alpha` Numeric(1), sets the false discovery rate threshold.
- `lfc` Numeric(1), sets the log fold change threshold.

Value

A list of 8 objects:

- `se` SummarizedExperiment object containing the original data
- `filt` SummarizedExperiment object containing the filtered data
- `norm` SummarizedExperiment object containing the normalized data
- `imputed` SummarizedExperiment object containing the imputed data
- `diff` SummarizedExperiment object containing FDR estimates of differential expression
dep SummarizedExperiment object annotated with logical columns indicating significant proteins
results data.frame containing all results variables from the performed analysis
param data.frame containing the test parameters

Examples

## Not run:
TMT_res <- TMT()

## End(Not run)

UbiLength - Ubiquitin interactors of different linear ubiquitin lengths
(UbIA-MS dataset)

Description

The UbiLength dataset contains label free quantification (LFQ) data for ubiquitin interactors of different linear ubiquitin lengths, generated by Zhang et al 2017. The dataset contains the protein-groups output file from MaxQuant.

Usage

UbiLength

Format

A data.frame with 3006 observations and 35 variables:

- **Protein.IDs** Uniprot IDs
- **Majority.protein.IDs** Uniprot IDs of major protein(s) in the protein group
- **Protein.names** Full protein names
- **Gene.names** Gene name
- **Fasta.headers** Header as present in the Uniprot fasta file
- **Peptides** Number of peptides identified for this protein group
- **Razor...unique.peptides** Number of peptides used for the quantification of this protein group
- **Unique.peptides** Number of peptides identified which are unique for this protein group
- **Intensity columns (12)** Raw mass spectrometry intensity, A.U.
- **LFQ.intensity columns (12)** LFQ normalized mass spectrometry intensity, A.U.
- **Only.identified.by.site** The protein is only identified by a modification site if marked (‘+’) 
- **Reverse** The protein is identified in the decoy database if marked (‘+’)
- **Potential.contaminant** The protein is a known contaminant if marked (‘+’)


**UbiLength_ExpDesign**

**Value**

A data.frame.

**Source**


---

**Description**

The `UbiLength_ExpDesign` object annotates 12 different samples of the UbiLength dataset in 4 conditions and 3 replicates.

**Usage**

`UbiLength_ExpDesign`

**Format**

A data.frame with 12 observations and 3 variables:

- **label**: Label names
- **condition**: Experimental conditions
- **replicate**: Replicate number

**Value**

A data.frame.

**Source**

Index

* datasets
  DiUbi, 7
  DiUbi_ExpDesign, 8
  UbiLength, 52
  UbiLength_ExpDesign, 53

  add_rejections, 3, 6, 13, 25–29, 31, 35, 39, 41–43, 48
  analyze_dep, 4, 6, 25
  as, 46

  daisy, 35
  DEP, 5, 49, 50
  DEP-package (DEP), 5
  DiUbi, 7, 7
  DiUbi_ExpDesign, 7, 8

  enrichr, 48

  filter_missval, 7, 9, 17, 22, 24, 47
  filter_proteins, 6, 10

  get_df_long, 7, 11
  get_df_wide, 7, 12
  get_prefix, 7, 13
  get_results, 6, 13
  get_suffix, 7, 14
  ggplot, 26–28, 30, 31, 33, 34, 36, 38–43

  Heatmap, 29, 32, 35, 37

  import_IsobarQuant, 6, 15, 44
  import_MaxQuant, 6, 16, 44
  impute, 6, 17, 18, 36, 44, 47, 51

  LFQ, 6, 18, 45

  make_se, 4, 6, 9–12, 17, 19, 22, 24, 38, 39, 44, 47
  make_se_parse, 4, 6, 9–12, 17, 20, 22, 24, 38, 39, 44, 47

  make_unique, 6, 19, 20, 21
  manual_impute, 7, 17, 18, 22, 44, 51
  meanSdPlot, 23, 23

  normalize_vsn, 6, 17, 22, 24, 36, 38, 47

  plot_all, 6, 24
  plot_cond, 7, 25
  plot_cond_freq, 7, 26
  plot_cond_overlap, 7, 27
  plot_cor, 6, 28
  plot_coverage, 6, 30
  plot_detect, 6, 30
  plot_dist, 31
  plot_frequency, 6, 32
  plot_gsea, 7, 33
  plot_heatmap, 6, 34
  plot_imputation, 6, 36
  plot_missval, 6, 37
  plot_normalization, 6, 37
  plot_numbers, 6, 38
  plot_p_hist, 7, 40
  plot_pca, 6, 39
  plot_single, 6, 41
  plot_volcano, 6, 42
  process, 6, 44

  report, 6, 45
  rmarkdown, 45
  run_app, 5, 45

  se2msn, 7, 46
  stat_binhex, 23

  test_diff, 3, 6, 13, 25–29, 31, 35, 39, 41–43, 47, 48
  test_gsea, 7, 33, 48
  theme_DEP1, 49
  theme_DEP2, 50
  TMT, 6, 43, 51

54
INDEX

UbiLength, 7, 52
UbiLength_ExpDesign, 7, 15, 16, 19, 53