Package 'protti'

May 10, 2021

Title Bottom-Up Proteomics and LiP-MS Quality Control and Data Analysis Tools

Version 0.1.1

Description Useful functions and workflows for proteomics quality control and data analysis of both limited proteolysis-coupled mass spectrometry (LiP-MS) (Feng et. al. (2014) <doi:10.1038/nbt.2999>) and regular bottom-up proteomics experiments. Data generated with search tools such as 'Spectronaut', 'MaxQuant' and 'Proteome Discover' can be easily used due to flexibility of functions.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

biocViews

Imports rlang, dplyr, stringr, magrittr, data.table, janitor, progress, purrr, tidyr, ggplot2, forcats, tibble, plotly, ggrepel, utils, grDevices, curl

RoxygenNote 7.1.1

Suggests testthat, covr, knitr, rmarkdown, proDA, limma, dendextend, pheatmap, heatmaply, viridis, iq, furrr, future, parallel, seriation, drc, naniar, httr, igraph, stringi, STRINGdb

Depends R (>= 4.0)

URL https://github.com/jpquast/protti, https://jpquast.github.io/protti/

BugReports https://github.com/jpquast/protti/issues

VignetteBuilder knitr

NeedsCompilation no

Author Jan-Philipp Quast [aut, cre], Dina Schuster [aut], ETH Zurich [cph, fnd]

Maintainer Jan-Philipp Quast <quast@imsb.biol.ethz.ch>

Repository CRAN

Date/Publication 2021-05-10 13:10:02 UTC

R topics documented:

`anova_protti` *Perform ANOVA*

Description

Performs an ANOVA statistical test

Usage

```
anova_protti(data, grouping, condition, mean_ratio, sd, n)
```

Arguments

`data`	A data frame containing at least the input variables.
`grouping`	The column in the data frame containing precursor or peptide identifiers.
`condition`	The column in the data frame containing the conditions.
`mean_ratio`	The column in the data frame containing mean intensities or mean intensity ratios.
`sd`	The column in the data frame containing the standard deviation corresponding to the mean.
`n`	The column in the data frame containing the number of replicates for which the corresponding mean was calculated.

Value

A data frame that contains the within group error (`ms_group`) and the between group error (`ms_error`), f statistic and p-values.

Examples

```
## Not run:
anova_protti(
  data,
  grouping = eg_precursor_id,
  condition = r_condition,
  mean = mean,
  sd = sd,
  n = n
)

## End(Not run)
```

`assign_missingness` *Assignment of missingness types*

Description

The type of missingness (missing at random, missing not at random) is assigned based on the comparison of a reference condition and every other condition.

Usage

```
assign_missingness(
  data,
  sample,
  condition,
  grouping,
  intensity,
  ref_condition = "control",
  completeness_MAR = 0.7,
  completeness_MNAR = 0.2,
  retain_columns = NULL
)
```

Arguments

`data`	A data frame containing at least the input variables.
`sample`	The column in the data data frame containing the sample name.
`condition`	The column in the data data frame containing the conditions.
`grouping`	The column in the data data frame containing precursor or peptide identifiers.
`intensity`	The column in the data data frame containing intensity values.
`ref_condition`	The condition that is used as a reference for missingness determination. By default `ref_condition = "control"`.

completeness_MAR
: The minimal degree of data completeness to be considered as MAR. Value has to be between 0 and 1, default is 0.7. It is multiplied with the number of replicates and then adjusted downward. The resulting number is the minimal number of observations for each condition to be considered as MAR. This number is always at least 1.

completeness_MNAR
: The maximal degree of data completeness to be considered as MNAR. Value has to be between 0 and 1, default is 0.20. It is multiplied with the number of replicates and then adjusted downward. The resulting number is the maximal number of observations for one condition to be considered as MNAR when the other condition is complete.

retain_columns
: A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns `retain_columns = NULL`. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

Value

A data frame that contains the reference condition paired with each treatment condition. The `comparison` column contains the comparison name for the specific treatment/reference pair. The `missingness` column reports the type of missingness.

- "complete": No missing values for every replicate of this reference/treatment pair for the specific grouping variable.
- "MNAR": Missing not at random. All replicates of either the reference or treatment condition have missing values for the specific grouping variable.
- "MAR": Missing at random. At least n-1 replicates have missing values for the reference/treatment pair for the specific grouping varible.

Examples

```
## Not run:
assign_missingness(
  data,
  sample = r_file_name,
  condition = r_condition,
  grouping = eg_precursor_id,
  intensity = normalised_intensity_log2,
  retain_columns = c(pg_protein_accessions)
)

## End(Not run)
```

barcode_plot *Barcode plot*

Description

Plots a "barcode plot" - a vertical line for each identified peptide. Peptides can be colored based on an additional variable. Also differential abundance can be displayed.

Usage

```
barcode_plot(
  data,
  start_position,
  end_position,
  protein_length,
  coverage = NULL,
  colouring = NULL,
  protein_id = NULL,
  facet = NULL,
  cutoffs = NULL
)
```

Arguments

data	Data frame containing differential abundance, start and end peptide or precursor positions and protein length.
start_position	Column in the data frame containing the start positions for each peptide or precursor.
end_position	Column in the data frame containing the end positions for each peptide or precursor.
protein_length	Column in the data frame containing the length of the protein.
coverage	Optional, column in the data frame containing coverage in percent. Will appear in the title of the barcode if provided.
colouring	Optional argument, column in the data frame containing information by which peptide or precursors should be colored.
protein_id	Optional argument, column in the data frame containing protein identifiers. Required if only one protein should be plotted and the data frame contains only information for this protein.
facet	Optional argument, column in the data frame containing information by which data should be faceted. This can be protein identifiers. Only 20 proteins are plotted at a time, the rest is ignored. If more should be plotted, a mapper over a subsetted data frame should be created.

cutoffs — Optional argument specifying the log2 fold change and significance cutoffs used for highlighting peptides. If this argument is provided colouring information will be overwritten with peptides that fulfill this condition. The cutoff should be provided in a vector of the form c(diff = 2, pval = 0.05). The name of the cutoff should reflect the column name that contains this information (log2 fold changes, p-values or adjusted p-values).

Value

A barcode plot is returned.

Examples

```
## Not run:
barcode_plot(
  data,
  start_position = start,
  end_position = end,
  protein_length = length,
  facet = pg_protein_accessions,
  cutoffs = c(diff = 2, pval = 0.05)
)

## End(Not run)
```

---

calculate_imputation *Sampling of values for imputation*

Description

`calculate_imputation` is a helper function that is used in the `impute` function. Depending on the type of missingness and method, it samples values from a normal distribution that can be used for the imputation. Note: The input intensities should be log2 transformed.

Usage

```
calculate_imputation(
  min = NULL,
  noise = NULL,
  mean = NULL,
  sd,
  missingness = c("MNAR", "MAR"),
  method = c("ludovic", "noise"),
  skip_log2_transform_error = FALSE
)
```

Arguments

min	minimal intensity value of the precursor/peptide. Is only required if method = "ludovic" and missingness = "MNAR".
noise	noise value for the precursor/peptide. Is only required if method = "noise" and missingness = "MNAR".
mean	mean intensity value of the condition with missing values for a given precursor/peptide. Is only required if missingness = "MAR".
sd	mean of the standard deviation of all conditions for a given precursor/peptide.
missingness	the missingness type of the data determines how values for imputation are sampled. This can be "MAR" or "MNAR".
method	the method to be used for imputation. For method = "ludovic", MNAR missingness is sampled around a value that is three lower (log2) than the lowest intensity value recorded for the precursor/peptide. For method = "noise", MNAR missingness is sampled around the noise value for the precursor/peptide.
skip_log2_transform_error	logical, if FALSE a check is performed to validate that input values are log2 transformed. If input values are > 40 the test is failed and an error is thrown.

Value

A vector of values for the imputation of missing data. The length of the vector depends on the number of replicates.

---

calculate_protein_abundance

Label-free protein quantification

Description

Determines relative protein abundances from ion quantification. Only proteins with at least 3 peptides are considered for quantification.

Usage

```
calculate_protein_abundance(
  data,
  sample,
  protein_id,
  precursor,
  peptide,
  intensity_log2,
  method = "iq",
  for_plot = FALSE,
  retain_columns = NULL
)
```

Arguments

data	A data frame that contains at least the input variables.
sample	The name of the column containing the sample name.
protein_id	The name of the column containing the protein accession numbers.
precursor	The name of the column containing precursors.
peptide	The name of the column containing peptide sequences. This column is needed to filter for proteins with at least 3 unique peptides. This can equate to more than three precursors. The quantification is done on the precursor level.
intensity_log2	The name of the column containing log2 transformed precursor intensities.
method	A character vector specifying with which method protein quantities should be calculated. Possible options include "sum", which takes the sum of all precursor intensities as the protein abundance. Another option is "iq", which performs protein quantification based on a maximal peptide ratio extraction algorithm that is adapted from the MaxLFQ algorithm of the MaxQuant software. Functions from the iq package are used. Default is "iq".
for_plot	A logical indicating whether the result should be only protein intensities or protein intensities together with precursor intensities that can be used for plotting using qc_protein_abundance. Default is FALSE.
retain_columns	A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns retain_columns = NULL. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

Value

If for_plot = FALSE, protein abundances are returned, if for_plot = TRUE also precursor intensities are returned. The later output is ideal for plotting with qc_protein_abundance and can be filtered to only include protein abundances.

Examples

```
## Not run:
calculate_protein_abundance(
  data,
  sample = r_file_name,
  protein_id = pg_protein_accessions,
  precursor = eg_precursor_id,
  peptide = pep_stripped_sequence,
  intensity_log2 = normalised_intensity_log2,
  method = "iq",
  retain_columns = c(pg_protein_accessions)
)

## End(Not run)
```

create_queue *Creates a mass spectrometer queue for Xcalibur*

Description

This function creates a measurement queue for sample acquisition for the software Xcalibur. All possible combinations of the provided information will be created to make file and sample names.

Usage

```
create_queue(
  date = NULL,
  instrument = NULL,
  user = NULL,
  measurement_type = NULL,
  experiment_name = NULL,
  digestion = NULL,
  treatment_type_1 = NULL,
  treatment_type_2 = NULL,
  treatment_dose_1 = NULL,
  treatment_dose_2 = NULL,
  treatment_unit_1 = NULL,
  treatment_unit_2 = NULL,
  n_replicates = NULL,
  number_runs = FALSE,
  organism = NULL,
  exclude_combinations = NULL,
  inj_vol = NA,
  data_path = NA,
  method_path = NA,
  position_row = NA,
  position_column = NA,
  blank_every_n = NULL,
  blank_position = NA,
  blank_method_path = NA,
  blank_inj_vol = 1,
  export = FALSE,
  export_to_queue = FALSE,
  queue_path = NULL
)
```

Arguments

date	Optional, start date of the measurements.
instrument	Optional, instrument initials.
user	Optional, user name.

`measurement_type`
: Optional, the measurement type of the samples (e.g "DIA", "DDA", "library" etc.).

`experiment_name`
: Optional, name of the experiment.

`digestion`
: Optional, the digestion types used in this experiment (e.g "LiP" and/or "tryptic control").

`treatment_type_1`
: Optional, name of the treatment.

`treatment_type_2`
: Optional, name of a second treatment that was combined with the first treatment.

`treatment_dose_1`
: Optional, doses used for treatment 1. These can be concentrations or times etc.

`treatment_dose_2`
: Optional, doses used for treatment 2. These can be concentrations or times etc.

`treatment_unit_1`
: Optional, unit of the doses for treatment 1 (e.g min, mM, etc.).

`treatment_unit_2`
: Optional, unit of the doses for treatment 2 (e.g min, mM, etc.).

`n_replicates`
: Optional, number of replicates used per sample.

`number_runs`
: Specifies if file names should be numbered from 1:n instead of adding experiment information. Default is FALSE.

`organism`
: Optional, name of the organism used.

`exclude_combinations`
: Optional, list of lists that contains vectors treatment types and treatment doses whos combinations should be excluded from the final queue.

`inj_vol`
: The volume used for injection in microliter. Will be `NA` if not specified. Then it needs to be manually specified before the queue can be used.

`data_path`
: The file path where the MS raw data should be saved. Backslashes should be escaped by another backslash. Will be `NA` if not specified, but needs to be specified later on then.

`method_path`
: The file path of the MS acquisition method. Backslashes should be escaped by another backslash. Will be `NA` if not specified, but needs to be specified later on then.

`position_row`
: The row positions that can be used for the samples (e.g c("A", "B")). If the number of specified rows and columns does not equal the total number of samples, positions will be repeated.

`position_column`
: The column positions that can be used for the samples (e.g 8). If the number of specified rows and columns does not equal the total number of samples, positions will be repeated.

`blank_every_n`
: Optional, specifies in which intervals a blank sample should be inserted.

`blank_position`
: The plate position of the blank. Will be `NA` if not specified, but needs to be specified later on then.

`blank_method_path`
: The file path of the MS acquisition method of the blank. Backslashes should be escaped by another backslash. Will be `NA` if not specified, but needs to be specified later on then.

`blank_inj_vol`
: Injection volume of the blank sample. Will be `NA` if not specified, but needs to be specified later on then.

`export`
: Logical, specifying if queue should be exported from R and saved as a .csv file. Default is TRUE. Further options for export can be adjusted with the `export_to_queue` and `queue_path` arguments.

`export_to_queue`
: Logical, specifying if the resulting queue should be appended to an already existing queue. If false result will be saved as `queue.csv`.

`queue_path`
: Optional, file path to a queue file to which the generated queue should be appended if `export_to_queue = TRUE`. If not specified queue file can be chosen interactively.

Value

If `export_to_queue = FALSE` a file named `queue.csv` will be returned that contains the generated queue. If `export_to_queue = TRUE`, the resulting generated queue will be appended to an already existing queue that needs to be specified either interactively or through the argument `queue_path`.

Examples

```
create_queue(
  date = c("200722"),
  instrument = c("EX1"),
  user = c("jquast"),
  measurement_type = c("DIA"),
  experiment_name = c("JPQ031"),
  digestion = c("LiP", "tryptic control"),
  treatment_type_1 = c("EDTA", "H2O"),
  treatment_type_2 = c("Zeba", "unfiltered"),
  treatment_dose_1 = c(10, 30, 60),
  treatment_unit_1 = c("min"),
  n_replicates = 4,
  number_runs = FALSE,
  organism = c("E. coli"),
  exclude_combinations = list(list(
    treatment_type_1 = c("H2O"),
    treatment_type_2 = c("Zeba", "unfiltered"),
    treatment_dose_1 = c(10, 30)
  )),
  inj_vol = c(2),
  data_path = "D:\\2007_Data",
  method_path = "C:\\Xcalibur\\methods\\user\\DIA_120min_41var_AGC200",
  position_row = c("A", "B", "C", "D", "E", "F"),
  position_column = 8,
  blank_every_n = 4,
  blank_position = "1-V1",
```

```
  blank_method_path = "C:\\Xcalibur\\methods\\blank"
)
```

`create_synthetic_data` *Creates a synthetic limited proteolysis proteomics dataset*

Description

This function creates a synthetic limited proteolysis proteomics dataset that can be used to test functions while knowing the ground truth.

Usage

```
create_synthetic_data(
  n_proteins,
  frac_change,
  n_replicates,
  n_conditions,
  method = "random_effect",
  concentrations = NULL,
  median_offset_sd = 0.05,
  mean_protein_intensity = 16.88,
  sd_protein_intensity = 1.4,
  mean_n_peptides = 12.75,
  size_n_peptides = 0.9,
  mean_sd_peptides = 1.7,
  sd_sd_peptides = 0.75,
  mean_log_replicates = -2.2,
  sd_log_replicates = 1.05,
  effect_sd = 2,
  dropout_curve_inflection = 14,
  dropout_curve_sd = -1.2,
  additional_metadata = TRUE
)
```

Arguments

`n_proteins`	Numeric, specifies the number of proteins in the synthetic dataset.
`frac_change`	Numeric, the fraction of proteins that has a peptide changing in abundance. So far only one peptide per protein is changing.
`n_replicates`	Numeric, the number of replicates per condition.
`n_conditions`	Numeric, the number of conditions.
`method`	Character, specifies the method type for the random sampling of significantly changing peptides. If `method = "random_effect"`, the effect for each condition is randomly sampled and conditions do not depend on each other. If `method = "dose_response"`, the effect is sampled based on a dose response curve and

conditions are related to each other depending on the curve shape. In this case the concentrations argument needs to be specified.

`concentrations` Numeric vector of the length of number of conditions, only needs to be specified if `method = "dose_response"`. This allows equal sampling of peptide intensities. It ensures that the same positions of dose response curves are sampled for each peptide based on the provided concentrations.

`median_offset_sd`
: Numeric, standard deviation of normal distribution that is used for sampling of inter-sample-differences. Default is 0.05.

`mean_protein_intensity`
: Numeric, mean of the protein intensity distribution. Default: 16.8.

`sd_protein_intensity`
: Numeric, standard deviation of the protein intensity distribution. Default: 1.4.

`mean_n_peptides`
: Numeric, mean number of peptides per protein. Default: 12.75.

`size_n_peptides`
: Numeric, dispersion parameter (the shape parameter of the gamma mixing distribution). Can be theoretically calculated as `mean + mean^2/variance`, however, it should be rather obtained by fitting the negative binomial distribution to real data. This can be done by using the `optim` function (see Example section). Default: 0.9.

`mean_sd_peptides`
: Numeric, mean of peptide intensity standard deviations within a protein. Default: 1.7.

`sd_sd_peptides` Numeric, standard deviation of peptide intensity standard deviation within a protein. Default: 0.75.

`mean_log_replicates, sd_log_replicates`
: Numeric, `meanlog` and `sdlog` value of the log normal distribution of replicate standard deviations. Can be obtained by fitting a log normal distribution to the distribution of replicate standard deviations from a real dataset. This can be done using the `optim` function (see Example section). Default: -2.2 and 1.05.

`effect_sd` Numeric, standard deviation of a normal distribution around `mean = 0` that is used to sample the effect of significantly changeing peptides. Default: 2.

`dropout_curve_inflection`
: Numeric, intensity inflection point of a probabilistic dropout curve that is used to sample intensity dependent missing values. This argument determines how many missing values there are in the dataset. Default: 14.

`dropout_curve_sd`
: Numeric, standard deviation of the probabilistic dropout curve. Needs to be negative to sample a droupout towards low intensities. Default: -1.2.

`additional_metadata`
: Logical, determines if metadata such as protein coverage, missed cleavages and charge state should be sampled and added to the list.

Value

A data frame that contains complete peptide intensities and peptide intensities with values that were created based on a probabilistic dropout curve.

Examples

```
create_synthetic_data(
  n_proteins = 10,
  frac_change = 0.1,
  n_replicates = 3,
  n_conditions = 2
)

# determination of mean_n_peptides and size_n_peptides parameters based on real data (count)
# example peptide count per protein
count <- c(6, 3, 2, 0, 1, 0, 1, 2, 2, 0)
theta <- c(mu = 1, k = 1)
negbinom <- function(theta) {
  -sum(stats::dnbinom(count, mu = theta[1], size = theta[2], log = TRUE))
}
fit <- stats::optim(theta, negbinom)
fit

# determination of mean_log_replicates and sd_log_replicates parameters
# based on real data (standard_deviations)

# example standard deviations of replicates
standard_deviations <- c(0.61, 0.54, 0.2, 1.2, 0.8, 0.3, 0.2, 0.6)
theta2 <- c(meanlog = 1, sdlog = 1)
lognorm <- function(theta2) {
 -sum(stats::dlnorm(standard_deviations, meanlog = theta2[1], sdlog = theta2[2], log = TRUE))
}
fit2 <- stats::optim(theta2, lognorm)
fit2
```

`diff_abundance` *Calculate differential abundance between conditions*

Description

Performs differential abundance calculations and statistical hypothesis tests on data frames with protein, peptide or precursor data. Different methods for statistical testing are available.

Usage

```
diff_abundance(
  data,
  sample,
  condition,
  grouping,
  intensity_log2,
  missingness,
  comparison,
  mean = NULL,
```

```
  sd = NULL,
  n_samples = NULL,
  ref_condition,
  filter_NA_missingness = TRUE,
  method = c("t-test", "t-test_mean_sd", "moderated_t-test", "proDA"),
  p_adj_method = "BH",
  retain_columns = NULL
)
```

Arguments

data	A data frame containing at least the input variables that are required for the selected method. Ideally the output of assign_missingness or impute is used.
sample	The column in the data frame containing the sample name. Is not required if method = "t-test_mean_sd".
condition	The column in the data frame containing the conditions.
grouping	The column in the data frame containing precursor or peptide identifiers.
intensity_log2	The column in the data frame containing intensity values. The intensity values need to be log2 transformed. Is not required if method = "t-test_mean_sd".
missingness	The column in the data frame containing missingness information. Can be obtained by calling assign_missingness. Is not required if method = "t-test_mean_sd".
comparison	The column in the data frame containing comparison information of treatment/reference condition pairs. Can be obtained by calling assign_missingness. Is not required if method = "t-test_mean_sd".
mean	The column in the data frame containing mean values for two conditions. Is only required if method = "t-test_mean_sd".
sd	The column in the data frame containing standard deviations for two conditions. Is only required if method = "t-test_mean_sd".
n_samples	The column in the data frame containing the number of samples per condition for two conditions. Is only required if method = "t-test_mean_sd".
ref_condition	The condition that is used as a reference for differential abundance calculation.
filter_NA_missingness	A logical, default is TRUE. For all methods except "t-test_mean_sd" missingness information has to be provided. If a reference/treatment pair has too few samples to be considered robust, it is annotated with NA as missingness. If this argument is TRUE, these reference/treatment pairs are filtered out.
method	A character vector, specifies the method used for statistical hypothesis testing. Methods include Welch test ("t-test"), a Welch test on means, standard deviations and number of replicates ("t-test_mean_sd") and a moderated t-test based on the limma package ("moderated_t-test"). More information on the moderated t-test can be found in the limma documentation. Furthermore, the proDA package specific method ("proDA") can be used to infer means across samples based on a probabilistic dropout model. This eliminates the need for data imputation since missing values are infered from the model. More information can be found in the proDA documentation.

p_adj_method A character vector, specifies the p-value correction method. Possible methods are c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"). Default method is "BH".

retain_columns A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns retain_columns = NULL. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

Value

A data frame that contains differential abundances (diff), p-values (pval) and adjusted p-values (adj_pval) for each protein, peptide or precursor (depending on the grouping variable) and the associated treatment/reference pair. Depending on the method the data frame contains additional columns:

- "t-test": The std_error column contains the standard error of the differential abundances. n_obs contains the number of observations for the specific protein, peptide or precursor (depending on the grouping variable) and the associated treatment/reference pair.
- "t-test_mean_sd": mean_control and mean_treated columns contain the means for the reference and treatment condition, respectively. sd_control and sd_treated columns contain the standard deviations for the reference and treatment condition, respectively. n_control and n_treated columns contain the numbers of samples for the reference and treatment condition, respectively. The std_error column contains the standard error of the differential abundances. t_statistic contains the t_statistic for the t-test.
- "moderated_t-test": CI_2.5 and CI_97.5 give the 2.5 contains average abundances for treatment/reference pairs (mean of the two group means). t_statistic contains the t_statistic for the t-test. B The B-statistic is the log-odds that the protein, peptide or precursor (depending on grouping) has a differential abundance between the two groups. Suppose B=1.5. The odds of differential abundance is exp(1.5)=4.48, i.e, about four and a half to one. The probability that there is a differential abundance is 4.48/(1+4.48)=0.82, i.e., the probability is about 82 abundant.n_obs contains the number of observations for the specific protein, peptide or precursor (depending on the grouping variable) and the associated treatment/reference pair.
- "proDA": The std_error column contains the standard error of the differential abundances. avg_abundance contains average abundances for treatment/reference pairs (mean of the two group means). t_statistic contains the t_statistic for the t-test. n_obs contains the number of observations for the specific protein, peptide or precursor (depending on the grouping variable) and the associated treatment/reference pair.

Examples

```
## Not run:
diff_abundance(
  data,
  sample = r_file_name,
  condition = r_condition,
  grouping = eg_precursor_id,
  intensity_log2 = normalised_intensity_log2,
  missingness = missingness,
  comparison = comparison,
```

```
  ref_condition = "control",
  method = "t-test",
  retain_columns = c(pg_protein_accessions)
)

## End(Not run)
```

`drc_4p` *Dose response curve helper function*

Description

This function peforms the four-parameter dose response curve fit. It is the helper function for the fit in the `fit_drc_4p` function.

Usage

```
drc_4p(data, response, dose, log_logarithmic = TRUE, pb = NULL)
```

Arguments

`data`	a data frame that contains at least the dose and response column the model should be fitted to.
`response`	the name of the column that contains the response values.
`dose`	the name of the column that contains the dose values.
`log_logarithmic`	logical indicating if a logarithmic or log-logarithmic model is fitted. If response values form a symmetric curve for non-log transformed dose values, a logarithmic model instead of a log-logarithmic model should be used. Usually biological dose response data has a log-logarithmic distribution, which is the reason this is the default. Log-logarithmic models are symmetric if dose values are log transformed.
`pb`	progress bar object. This is only necessary if the function is used in an iteration.

Value

An object of class `drc`. If no fit was performed a character vector with content "no_fit".

`extract_metal_binders` *Extract metal-bind protein information from UniProt*

Description

Information of metal binding proteins is extracted from UniProt data retrieved with `fetch_uniprot`. ChEBI IDs, potential sub-IDs for metal cations, binding site locations in the protein and sub-ID evidence level (based on metal presence as cofactor) are extracted.

Usage

```
extract_metal_binders(
  data,
  protein_id = id,
  feature_metal_binding = feature_metal_binding,
  chebi_cofactor = chebi_cofactor,
  chebi_catalytic_activity = chebi_catalytic_activity,
  chebi_data = NULL,
  chebi_relation_data = NULL
)
```

Arguments

`data`	A data frame containing at least the input columns.
`protein_id`	The name of the column containing protein identifiers.
`feature_metal_binding`	The name of the column containing feature metal binding information from UniProt.
`chebi_cofactor`	The name of the column containing ChEBI cofactor information from UniProt.
`chebi_catalytic_activity`	The name of the column containing ChEBI catalytic activity information from UniProt.
`chebi_data`	Optional, a data frame that can be manually obtained with `fetch_chebi()`. If not provided it will be fetched within the function. If the function is run many times it is recommended to provide the data frame to save time.
`chebi_relation_data`	Optional, a data frame that can be manually obtained with `fetch_chebi(relation = TRUE)`. If not provided it will be fetched within the function. If the function is run many times it is recommended to provide the data frame to save time.

Value

A data frame containing information on protein metal binding state. It contains the following types of columns (the naming might vary based on the input):

- `protein_id`: UniProt protein identifier.

- source: The source of the information, can be either feature_metal_binding, chebi_cofactor or chebi_catalytic_activity.
- ids: ChEBI ID assigned to protein and binding site based on metal_type column name. These are general IDs that have sub-IDs. Thus, they generally describe the type of metal ion bound to the protein.
- metal_position: Amino acid position within the protein that is involved in metal binding.
- metal_type: Metal name extracted from feature_metal_binding information. This is the name that is used as a search pattern in order to assign a ChEBI ID with the split_metal_name helper function within this function.
- sub_ids: ChEBI ID that is a sub-ID (incoming) of the ID in the ids column. Thus, they more specifically describe the potential nature of the metal ion.
- main_id_name: Official ChEBI name associated with the ID in the ids column.
- multi_evidence: If there is overlapping information in feature_metal_binding and chebi_cofactor or chebi_catalytic_activity, only feature_metal_binding is retained and multi_evidence is TRUE.
- sub_id_name: Official ChEBI name associated with the ID in the sub_ids column.

Examples

```
## Not run:
extract_metal_binders(
  data,
  protein_id = id,
  feature_metal_binding = feature_metal_binding,
  chebi_cofactor = chebi_cofactor,
  chebi_catalytic_activity = chebi_catalytic_activity
)

## End(Not run)
```

---

fetch_chebi *Fetch ChEBI database information*

Description

Fetches all information from the ChEBI database.

Usage

```
fetch_chebi(relation = FALSE)
```

Arguments

relation	a logical, if TRUE, ChEBI Ontology data will be returned instead the main compound data. This data can be used to check the relations of ChEBI ID's to eachother.

Value

A data frame that contains all information about each molecule in the ChEBI database. Only "3-star" observations are included in the result. These are entries manually annotated by the ChEBI curator team.

Examples

```
head(fetch_chebi())
```

`fetch_go` *Fetch gene ontology information from geneontology.org*

Description

Fetches gene ontology data from geneontology.org for the provided organism ID.

Usage

```
fetch_go(organism_id)
```

Arguments

`organism_id`	An NCBI taxonomy identifier of an organism (TaxId). Possible inputs inlude only: "9606" (Human), "559292" (Yeast) and "83333" (E. coli).

Value

A data frame that contains gene ontology mappings to UniProt or SGD IDs. The original file is a .GAF file. A detailed description of all columns can be found here: http://geneontology.org/docs/go-annotation-file-gaf-format-2.1/

Examples

```
fetch_go("9606")
```

fetch_kegg *Fetch KEGG pathway data from KEGG*

Description

Fetches gene IDs and corresponding pathway IDs and names for the provided organism.

Usage

```
fetch_kegg(species)
```

Arguments

species	a character vector providing an abreviated species name. "hsa" for human, "eco" for E. coli and "sce" for S. cerevisiae. Additional possible names can be found for eukaryotes and for prokaryotes.

Value

A data frame that contains gene IDs with corresponding pathway IDs and names for a selected organism.

Examples

```
head(fetch_kegg(species = "hsa"))
```

fetch_mobidb *Fetch protein disorder information from MobiDB*

Description

Fetches information about disordered protein regions from MobiDB.

Usage

```
fetch_mobidb(organism_id, protein_ids)
```

Arguments

organism_id	An NCBI taxonomy identifier of an organism (TaxId). Possible inputs inlude only: "9606" (Human), "559292" (Yeast), "83333" (E. coli), "10090" (Mouse), "9913" (Bovine), "7227" (Fruit fly).
protein_ids	A character vector of UniProt identifiers. These need to be proteins from the organism provided in organism_id.

Value

A data frame that contains start and end positions for disordered regions for each protein provided. The `feature` column contains information on the source of this annotation. More information on the source can be found here.

Examples

```
fetch_mobidb(
  organism_id = "83333",
  protein_ids = c("P36578", "O43324", "Q00796")
)
```

`fetch_uniprot` *Fetch protein data from UniProt*

Description

Fetches protein metadata from UniProt.

Usage

```
fetch_uniprot(
  uniprot_ids,
  columns = c("protein names", "length", "sequence", "genes", "database(GeneID)",
    "database(String)", "go(molecular function)", "go(biological process)",
    "go(cellular compartment)", "interactor", "feature(ACTIVE SITE)",
    "feature(BINDING SITE)", "feature(METAL BINDING)", "chebi(Cofactor)",
    "chebi(Catalytic activity)", "database(PDB)"),
  batchsize = 200,
  show_progress = TRUE
)
```

Arguments

`uniprot_ids`	A character vector of UniProt accession numbers
`columns`	Metadata columns that should be imported from UniProt (all possible columns can be found here.)
`batchsize`	Size of batch of proteins for a single query
`show_progress`	Logical, if true, a progress bar will be shown

Value

A data frame that contains all protein metadata specified in `columns` for the proteins provided. If an invalid ID was provided that contains a valid UniProt ID, the valid portion of the ID is fetched and the invalid input ID is saved in a column called `input_id`.

Examples

```
fetch_uniprot(c("P36578", "O43324", "Q00796"))
```

`fetch_uniprot_proteome` *Fetch proteome data from UniProt*

Description

Fetches proteome data from UniProt for the provided organism ID.

Usage

```
fetch_uniprot_proteome(organism_id, columns = c("id"), reviewed = TRUE)
```

Arguments

`organism_id`	a numeric vector of one NCBI taxonomy identifier (TaxId) for and organism of choice.
`columns`	a character vector of metadata columns that should be imported from UniProt (all possible columns can be found here: https://www.uniprot.org/help/uniprotkb_column_names). Note: Not more than one or two columns should be selected otherwise the function will not be able to efficiently retrieve the information. If more information is needed, `fetch_uniprot()` can be used with the IDs retrieved by this function.
`reviewed`	a logical. If true, only reviewed protein entries will be retrieved.

Value

A data frame that contains all protein metadata specified in `columns` for the organism of choice.

Examples

```
head(fetch_uniprot_proteome(9606))
```

`filter_cv` *Data filtering based on coefficients of variation (CV)*

Description

Filters the input data based on precursor, peptide or protein intensity coefficients of variation. The function should be used to ensure that only robust measurements and quantifications are used for data analysis. It is advised to use the function after inspection of raw values (quality control) and median normalisation. Generally, the function calculates CVs of each peptide, precursor or protein for each condition and removes peptides, precursors or proteins that have a CV above the cutoff in less than the (user-defined) required number of conditions. Since the user-defined cutoff is fixed and does not depend on the number of conditions that have detected values, the function might bias for data completeness.

Usage

```
filter_cv(
  data,
  grouping,
  condition,
  log2_intensity,
  cv_limit = 0.25,
  min_conditions,
  silent = FALSE
)
```

Arguments

`data`	Data frame containing at least the input variables.
`grouping`	Column in the data frame containing the grouping variable that can be either precursors, peptides or proteins.
`condition`	Column in the data frame containing information on the sample condition.
`log2_intensity`	Column in the data frame containing log2 transformed intensities.
`cv_limit`	Optional argument specifying the CV cutoff that will be applied. Default is 0.25.
`min_conditions`	The minimum number of conditions for which grouping CVs should be below the cutoff.
`silent`	Logical argument specifiying if a message with the number of filtered out conditions should be returned.

Value

The CV filtered data frame.

Examples

```
## Not run:
filter_cv(
  data,
  grouping = eg_precursor_id,
  condition = r_condition,
  log2_intensity = normalised_intensity_log2,
  cv_limit = 0.25,
  min_conditions = 5
)

## End(Not run)
```

`find_all_subs` *Find all ChEBI sub IDs of an ID*

Description

For a given ChEBI ID, find all ChEBI sub IDs (incoming IDs) and their sub IDs. The type of relationship can be selected too. This is a helper function for other functions.

Usage

```
find_all_subs(data, id, type = "is_a")
```

Arguments

`data`	A data frame that contains information on ChEBI IDs (id), their sub IDs (incoming) and their relationship (type). This data frame can be obtained by calling `fetch_chebi(relation = TRUE)`.
`id`	A character vector of ChEBI IDs for which sub IDs should be retreived.
`type`	A character vector containing the type of relationship that should be considered for retreival. It is possible to use "all" relationships. The default type is "is_a". A list of possible relationships can be found here.

Value

A list of character vector containing the provided ID and all of its sub IDs. It contains one element per input ID.

find_chebis *Find ChEBI IDs for name patterns*

Description

A list of ChEBI IDs that contain a specific name pattern is returned.

Usage

```
find_chebis(chebi_data, pattern)
```

Arguments

chebi_data	A data frame that contains at least information on ChEBI IDs (id) and their names (name). This data frame can be obtained by calling `fetch_chebi()`. Ideally this should be subsetted to only contain molecules of a specific type e.g. metals. This can be achieved by calling `find_all_subs` with a general ID such as "25213" (Metal cation) and then subset the complete ChEBI database to only include the returned sub-IDs. Using a subsetted database ensures better search results. This is a helper function for other functions.
pattern	A character vector that contains names or name patterns of molecules. Name patterns can be for example obtained with the `split_metal_name` function.

Value

A list of character vectors containing ChEBI IDs that have a name matching the supplied pattern.

find_peptide *Find peptide location*

Description

The position of the given peptide sequence is searched within the given protein sequence. In addition the last amino acid of the peptide and the amino acid right before are reported.

Usage

```
find_peptide(data, protein_sequence, peptide_sequence)
```

Arguments

data	A data frame containing at least the protein and peptide sequence.
protein_sequence	The name of the column containing the protein sequence.
peptide_sequence	The name of the column containing the peptide sequence.

Value

A data frame that contains the input data and four additional columns with start and end position, the last amino acid and the amino acid before.

Examples

```
data <- data.frame(protein_sequence = c("abcdefg"), peptide_sequence = c("cde"))
find_peptide(data, protein_sequence, peptide_sequence)
```

`fit_drc_4p` *Fitting four-parameter dose response curves*

Description

Function for fitting four-parameter dose response curves for each group (precursor, peptide or protein). In addition it can filter data based on completeness, the completeness distribution and statistical testing using ANOVA.

Usage

```
fit_drc_4p(
  data,
  sample,
  grouping,
  response,
  dose,
  filter = "post",
  replicate_completeness = 0.7,
  condition_completeness = 0.5,
  correlation_cutoff = 0.8,
  log_logarithmic = TRUE,
  include_models = FALSE,
  retain_columns = NULL
)
```

Arguments

`data`	A data frame containing at least the input variables.
`sample`	The name of the column containing the sample names.
`grouping`	The name of the column containing precursor, peptide or protein identifiers.
`response`	The name of the column containing response values, eg. log2 transformed intensities.
`dose`	The name of the column containing dose values, eg. the treatment concentrations.

`filter` A character vector indicating if models should be filtered and if they should be filtered before or after the curve fits. Filtering of models can be skipped with `filter = "none"`. Data can be filtered prior to model fitting with `filter = "pre"`. In that case models will only be fitted for data that passed the filtering step. This will allow for faster model fitting since only fewer models will be fit. If you plan on performing an enrichment analysis you have to choose `filter = "post"`. All models will be fit (even the ones that do not pass the filtering criteria). For enrichment analysis you should use both good (i.e. models that pass the filtering) and bad (i.e. models that do not pass the filtering) models. Therefore, for post-filtering the full list is returned and it will only contain annotations that indicate (`passed_filter`) if the filtering was passed or not. Default is "post". For ANOVA an adjusted p-value of 0.05 is used as a cutoff.

`replicate_completeness`
Similar to `completenss_MAR` of the `assign_missingness` function this argument sets a threshold for the completeness of data. In contrast to `assign_missingness` it only determines the completeness for one condition and not the comparison of two conditions. The threshold is used to calculate a minimal degree of data completeness. The value provided to this argument has to be between 0 and 1, default is 0.7. It is multiplied with the number of replicates and then adjusted downward. The resulting number is the minimal number of observations that a condition needs to have to be considered "complete enough" for the `condition_completeness` argument.

`condition_completeness`
This argument determines how many conditions need to at least fulfill the "complete enough" criteria set with `replicate_completeness`. The value provided to this argument has to be between 0 and 1, default is 0.5. It is multiplied with the number of conditions and then adjusted downward. The resulting number is the minimal number of conditions that need to fulfill the `replicate_completeness` argument for a peptide to pass the filtering.

`correlation_cutoff`
A numeric vector specifying the correlation cutoff used for data filtering.

`log_logarithmic`
logical indicating if a logarithmic or log-logarithmic model is fitted. If response values form a symmetric curve for non-log transformed dose values, a logarithmic model instead of a log-logarithmic model should be used. Usually biological dose response data has a log-logarithmic distribution, which is the reason this is the default. Log-logarithmic models are symmetric if dose values are log transformed.

`include_models` A logical indicating if model fit objects should be exported. These are usually very large and not necessary for further analysis.

`retain_columns` A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns `retain_columns = NULL`. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

Details

If data filtering options are selected, data is filtered based on multiple criteria. In general, curves are only fitted if there are at least 5 conditions with data points present to ensure that there is potential for a good curve fit. Therefore, this is also the case if no filtering option is selected. Furthermore, a completeness cutoff is defined for filtering. By default each entity (e.g. precursor) is filtered to contain at least 70 all conditions (adjusted downward). This can be adjusted with the according arguments. In addition to the completeness cutoff, also a significance cutoff is applied. ANOVA is used to compute the statistical significance of the change for each entity. The resulting p-value is adjusted using the Benjamini-Hochberg method and a cutoff of q <= 0.05 is applied. Curve fits that have a minimal value that is higher than the maximal value are excluded as they were likely wrongly fitted. Curves with a correlation below 0.8 are not passing the filtering. If a fit does not fulfill the significance or completeness cutoff, it has a chance to still be considered if half of its values (+/-1 value) pass the replicate completeness criteria and half do not pass it. In order to fall into this category, the values that fulfill the completeness cutoff and the ones that do not fulfill it need to be consecutive, meaning located next to each other based on their concentration values. Furthermore, the values that do not pass the completeness cutoff need to be lower in intensity. Lastly, the difference between the two groups is tested for statistical significance using a Welch's t-test and a cutoff of p <= 0.1 (we want to mainly discard curves that falsly fit the other criteria but that have clearly non-significant differences in mean). This allows curves to be considered that have missing values in half of their observations due to a decrease in intensity. It can be thought of as conditions that are missing not at random (MNAR). It is often the case that those entities do not have a significant p-value since half of their conditions are not considered due to data missingness.

The final filtered list is ranked based on a score calculated on entities that pass the filter. The score is the negative log10 of the adjusted ANOVA p-value scaled between 0 and 1 and the correlation scaled between 0 and 1 summed up and divided by 2. Thus, the highest score an entity can have is 1 with both the highest correlation and adjusted p-value. The rank is corresponding to this score. Please note, that entities with MNAR conditions might have a lower score due to the missing or non-significant ANOVA p-value. You should have a look at curves that are TRUE for `dose_MNAR` in more detail.

Value

If `include_models = FALSE` a data frame is returned that contains correlations of predicted to measured values as a measure of the goodness of the curve fit, an associated p-value and the four parameters of the model for each group. Furthermore, input data for plots is returned in the columns `plot_curve` (curve and confidence interval) and `plot_points` (measured points). If \ codeinclude_models = TURE, a list is returned that contains:

- `fit_objects`: The fit objects of type `drc` for each group.
- `correlations`: The correlation data frame described above

Examples

```
## Not run:
fit_drc_4p(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  response = intensity,
```

```
  dose = concentration
)

## End(Not run)
```

`go_enrichment` *Perform gene ontology enrichment analysis*

Description

Analyses enrichment of gene ontology terms associated with proteins in the fraction of significant proteins compared to all detected proteins. A two-sided Fisher's exact test is performed to test significance of enrichment or depletion. GO annotations can be provided to this function either through UniProt `go_annotations_uniprot`, through a table obtained with `fetch_go` in the `go_data` argument or GO annotations are fetched automatically by the function by providing `ontology_type` and `organism_id`.

Usage

```
go_enrichment(
  data,
  protein_id,
  is_significant,
  go_annotations_uniprot = NULL,
  ontology_type,
  organism_id = NULL,
  go_data = NULL,
  plot = TRUE,
  plot_cutoff = "adj_pval top10"
)
```

Arguments

`data`	A data frame that contains at least the input variables.
`protein_id`	The name of the column containing the protein accession numbers.
`is_significant`	The name of the column containing a logical indicating if the corresponding protein has a significantly changing peptide. The input data frame may contain peptide level information with significance information. The function is able to extract protein level information from this.
`go_annotations_uniprot`	(Recommended) The name of the column containing gene ontology annotations obtained from UniProt using `fetch_uniprot`. These annotations are already separated into the desired ontology type so the argument `ontology_type` is not required.

ontology_type	Optional, A character vector specifying the type of ontology that should be used. Possible values are molecular function (MF), biological process (BP), cellular component (CC). This argument is not required if GO annotations are provided from UniProt in go_annotations_uniprot. It is required if annotations are provided through go_data or automatically fetched.
organism_id	Optional, An NCBI taxonomy identifier of an organism (TaxId). Possible inputs include only: "9606" (Human), "559292" (Yeast) and "83333" (E. coli). Is only necessary if GO data is not provided either by go_annotations_uniprot or in go_data.
go_data	Optional, a data frame that can be obtained with fetch_go. If you provide data yourself make sure column names for protein ID (db_id) and GO ID (go_id) are the same as for data obtained with fetch_go.
plot	A logical indicating whether the result should be plotted or returned as a table.
plot_cutoff	A character vector indicating if the plot should contain the top 10 most significant proteins (p-value or adjusted p-value), or if a significance cutoff should be used to determine the number of GO terms in the plot. This information should be provided with the type first followed by the threshold separated by a space. Example are plot_cutoff = "adj_pval top10", plot_cutoff = "pval 0.05" or plot_cutoff = "adj_pval 0.01". The threshold can be chosen freely.

Value

A bar plot displaying negative log10 adjusted p-values for the top 10 enriched or depleted gene ontology terms. Alternatively, plot cutoffs can be chosen individually with the plot_cutoff argument. Bars are colored according to the direction of the enrichment. If plot = FALSE, a data frame is returned. P-values are adjusted with Benjamini-Hochberg.

Examples

```
## Not run:
go_enrichment(
  data,
  protein_id = pg_protein_accessions,
  is_significant = significant,
  go_annotations_uniprot = go_molecular_function
)

## End(Not run)
```

---

impute *Imputation of missing values*

Description

impute is calculating imputation values for missing data depending on the selected method.

Usage

```
impute(
  data,
  sample,
  grouping,
  intensity,
  condition,
  comparison,
  missingness,
  noise = NULL,
  method,
  skip_log2_transform_error = FALSE,
  retain_columns = NULL
)
```

Arguments

data	a dataframe that is ideally the output from the assign_missingness function. It should containing at least the input variables. For each "reference_vs_treatment" comparison, there should be the pair of the reference and treatment condition. That means the reference condition should be doublicated once for every treatment.
sample	the name of the column containing the sample names.
grouping	the name of the column containing precursor or peptide identifiers.
intensity	the name of the column containing intensity values. Note: The input intensities should be log2 transformed.
condition	the name of the column containing the conditions.
comparison	the name of the column containing the comparisons of treatment/reference pairs. This is an output of the assign_missingnes function.
missingness	the name of the column that contains the missingness type of the data determines how values for imputation are sampled. This should at least contain "MAR" or "MNAR".
noise	the name of the column that contains the noise value for the precursor/peptide. Is only required if method = "noise". Note: Noise values need to be log2 transformed.
method	the method to be used for imputation. For method = "ludovic", MNAR missingness is sampled from a normal distribution around a value that is three lower (log2) than the lowest intensity value recorded for the precursor/peptide and that has a spread of the mean standard deviation for the precursor/peptide. For method = "noise", MNAR missingness is sampled from a normal distribution around the mean noise for the precursor/peptide and that has a spread of the mean standard deviation (from each condition) for the precursor/peptide.
skip_log2_transform_error	logical, if FALSE a check is performed to validate that input values are log2 transformed. If input values are > 40 the test is failed and an error is thrown.

`retain_columns` A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns `retain_columns = NULL`. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

Value

A data frame that contains an `imputed_intensity` and `imputed` column in addition to the required input columns. The `imputed` column indicates if a value was imputed. The `imputed_intensity` column contains imputed intensity values for previously missing intensities.

Examples

```
## Not run:
impute(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity = intensity_log2,
  condition = r_condition,
  comparison = comparison,
  missingness = missingness,
  method = "ludovic"
)

## End(Not run)
```

---

kegg_enrichment *Perform KEGG pathway enrichment analysis*

Description

Analyses enrichment of KEGG pathways associated with proteins in the fraction of significant proteins compared to all detected proteins. A Fisher's exact test is performed to test significance of enrichment.

Usage

```
kegg_enrichment(
  data,
  protein_id,
  is_significant,
  pathway_id = pathway_id,
  pathway_name = pathway_name,
  plot = TRUE,
  plot_cutoff = "adj_pval top10"
)
```

Arguments

data	A data frame that contains at least the input variables.
protein_id	The name of the column containing the protein accession numbers.
is_significant	The name of the column containing a logical indicating if the corresponding protein has a significantly changing peptide. The input data frame may contain peptide level information with significance information. The function is able to extract protein level information from this.
pathway_id	The name of the column containing KEGG pathway identifiers. These can be obtained from KEGG using fetch_kegg.
pathway_name	The name of the column containing KEGG pathway names. These can be obtained from KEGG using fetch_kegg.
plot	A logical indicating whether the result should be plotted or returned as a table.
plot_cutoff	A character vector indicating if the plot should contain the top 10 most significant proteins (p-value or adjusted p-value), or if a significance cutoff should be used to determine the number of GO terms in the plot. This information should be provided with the type first followed by the threshold separated by a space. Example are plot_cutoff = "adj_pval top10", plot_cutoff = "pval 0.05" or plot_cutoff = "adj_pval 0.01". The threshold can be chosen freely.

Value

A bar plot displaying negative log10 adjusted p-values for the top 10 enriched pathways. Bars are coloured according to the direction of the enrichment. If plot = FALSE, a data frame is returned.

Examples

```
## Not run:
kegg_enrichment(
  data,
  protein_id = pg_protein_accessions,
  is_significant = significant,
  pathway_id = pathway_id,
  pathway_name = pathway_name
)

## End(Not run)
```

---

median_normalisation *Median normalisation*

Description

Performs median normalisation on intensities. The normalised intensity is the original intensity minus the run median plus the global median. This is also the way it is implemented in Spectronaut.

Usage

```
median_normalisation(data, sample, intensity_log2)
```

Arguments

data	A data frame containing at least sample names and intensity values.
sample	The name of the column containing the sample names.
intensity_log2	The name of the column containing the log2 transformed intensity values to be normalised.

Value

A dataframe with a column called `normalised_intensity_log2` containing the normalised intensity values.

Examples

```
data <- data.frame(
  r_file_name = c("s1", "s2", "s3", "s1", "s2", "s3"),
  intensity_log2 = c(18, 19, 17, 20, 21, 19)
)

median_normalisation(data,
  sample = r_file_name,
  intensity_log2 = intensity_log2
)
```

---

network_analysis *Analyse protein interaction network for significant hits*

Description

The STRING database provides a resource for known and predicted protein-protein interactions. The type of interactions include direct (physical) and indirect (functional) interactions. Through the R package `STRINGdb` this resource if provided to R users. This function provides a convenient wrapper for `STRINGdb` functions that allow an easy use within the protti pipeline.

Usage

```
network_analysis(
  data,
  protein_id,
  string_id,
  organism_id,
  score_threshold = 900,
  binds_treatment = NULL,
  halo_color = NULL,
  plot = TRUE
)
```

Arguments

`data`	A data frame that contains significantly changing proteins (STRINGdb is only able to plot 400 proteins at a time so do not provide more for network plots). Information about treatment binding can be provided and will be displayed as colorful halos around the proteins in the network.
`protein_id`	The name of the column containing the protein accession numbers.
`string_id`	The name of the column containing STRING database identifiers. These can be obtained from UniProt.
`organism_id`	Numeric organism ID (NCBI taxon-ID). This can be obtained from here. H. sapiens: 9606, S. cerevisiae: 4932, E. coli: 511145.
`score_threshold`	The interaction score based on STRING has to be between 0 and 1000. A score closer to 1000 is related to a higher confidence for the interaction. The default value is 900.
`binds_treatment`	The name of the column containing a logical indicating if the corresponding protein binds to the treatment. This information can be obtained from different databases, e.g UniProt.
`halo_color`	Optional, A character vector with a color hex-code. This is the color of the halo of proteins that bind the treatment.
`plot`	A logical indicating whether the result should be plotted or returned as a table.

Value

A network plot displaying interactions of the provided proteins. If `binds_treatment` was provided halos around the proteins show which proteins interact with the treatment. If `plot = FALSE` a table with interaction information is returned.

Examples

```
## Not run:
network_analysis(
  data,
  protein_id = pg_protein_accessions,
  string_id = database_string,
  organism_id = 511145,
  binds_treatment = is_known,
  plot = TRUE
)

## End(Not run)
```

`parallel_fit_drc_4p` *Fitting four-parameter dose response curves (using parallel processing)*

Description

This function is a wrapper around `fit_drc_4p` that allows the use of all system cores for model fitting. It should only be used on systems that have enough memory available. Workers can either be set up manually before running the function with `future::plan(multiprocess)` or automatically by the function (maximum number of workers is 12 in this case). If workers are set up manually the number of cores should be provided to `n_cores`. Worker can be terminated after completion with `future::plan(sequential)`. It is not possible to export the individual fit objects when using this function as compared to the non parallel function as they are too large for efficient export from the workers.

Usage

```
parallel_fit_drc_4p(
  data,
  sample,
  grouping,
  response,
  dose,
  filter = "post",
  replicate_completeness = 0.7,
  condition_completeness = 0.5,
  correlation_cutoff = 0.8,
  log_logarithmic = TRUE,
  retain_columns = NULL,
  n_cores = NULL
)
```

Arguments

`data`	A data frame containing at least the input variables.
`sample`	The name of the column containing the sample names.
`grouping`	The name of the column containing precursor, peptide or protein identifiers.
`response`	The name of the column containing response values, eg. log2 transformed intensities.
`dose`	The name of the column containing dose values, eg. the treatment concentrations.
`filter`	A character vector indicating if models should be filtered. The option `"pre"` is not available for parallel fitting of models. This is because ANOVA adjusted p-values would be wrongly calculated because the dataset is split onto multiple

cores. Default is "post" and we recommend always using "post" because compared to "none" only some additional columns are added that contain the filter information. For ANOVA an adjusted p-value of 0.05 is used as a cutoff.

`replicate_completeness`
: Similar to `completenss_MAR` of the `assign_missingness` function this argument sets a threshold for the completeness of data. In contrast to `assign_missingness` it only determines the completeness for one condition and not the comparison of two conditions. The threshold is used to calculate a minimal degree of data completeness. The value provided to this argument has to be between 0 and 1, default is 0.7. It is multiplied with the number of replicates and then adjusted downward. The resulting number is the minimal number of observations that a condition needs to have to be considered "complete enough" for the `condition_completeness` argument.

`condition_completeness`
: This argument determines how many conditions need to at least fulfill the "complete enough" criteria set with `replicate_completeness`. The value provided to this argument has to be between 0 and 1, default is 0.5. It is multiplied with the number of conditions and then adjusted downward. The resulting number is the minimal number of conditions that need to fulfill the `replicate_completeness` argument for a peptide to pass the filtering.

`correlation_cutoff`
: A numeric vector specifying the correlation cutoff used for data filtering.

`log_logarithmic`
: logical indicating if a logarithmic or log-logarithmic model is fitted. If response values form a symmetric curve for non-log transformed dose values, a logarithmic model instead of a log-logarithmic model should be used. Usually biological dose response data has a log-logarithmic distribution, which is the reason this is the default. Log-logarithmic models are symmetric if dose values are log transformed.

`retain_columns`
: A vector indicating if certain columns should be retained from the input data frame. Default is not retaining additional columns `retain_columns = NULL`. Specific columns can be retained by providing their names (not in quotations marks, just like other column names, but in a vector).

`n_cores`
: Optional, the number of cores used if workers are set up manually.

Details

If data filtering options are selected, data is filtered based on multiple criteria. In general, curves are only fitted if there are at least 5 conditions with data points present to ensure that there is potential for a good curve fit. Therefore, this is also the case if no filtering option is selected. Furthermore, a completeness cutoff is defined for filtering. By default each entity (e.g. precursor) is filtered to contain at least 70 all conditions (adjusted downward). This can be adjusted with the according arguments. In addition to the completeness cutoff, also a significance cutoff is applied. ANOVA is used to compute the statistical significance of the change for each entity. The resulting p-value is adjusted using the Benjamini-Hochberg method and a cutoff of q <= 0.05 is applied. Curve fits that have a minimal value that is higher than the maximal value are excluded as they were likely wrongly fitted. Curves with a correlation below 0.8 are not passing the filtering. If a fit does not

fulfill the significance or completeness cutoff, it has a chance to still be considered if half of its values (+/-1 value) pass the replicate completeness criteria and half do not pass it. In order to fall into this category, the values that fulfill the completeness cutoff and the ones that do not fulfill it need to be consecutive, meaning located next to each other based on their concentration values. Furthermore, the values that do not pass the completeness cutoff need to be lower in intensity. Lastly, the difference between the two groups is tested for statistical significance using a Welch's t-test and a cutoff of p <= 0.1 (we want to mainly discard curves that falsly fit the other criteria but that have clearly non-significant differences in mean). This allows curves to be considered that have missing values in half of their observations due to a decrease in intensity. It can be thought of as conditions that are missing not at random (MNAR). It is often the case that those entities do not have a significant p-value since half of their conditions are not considered due to data missingness.

The final filtered list is ranked based on a score calculated on entities that pass the filter. The score is the negative log10 of the adjusted ANOVA p-value scaled between 0 and 1 and the correlation scaled between 0 and 1 summed up and divided by 2. Thus, the highest score an entity can have is 1 with both the highest correlation and adjusted p-value. The rank is corresponding to this score. Please note, that entities with MNAR conditions might have a lower score due to the missing or non-significant ANOVA p-value. You should have a look at curves that are TRUE for `dose_MNAR` in more detail.

Value

A data frame is returned that contains correlations of predicted to measured values as a measure of the goodness of the curve fit, an associated p-value and the four parameters of the model for each group. Furthermore, input data for plots is returned in the columns `plot_curve` (curve and confidence interval) and `plot_points` (measured points).

Examples

```
## Not run:
parallel_fit_drc_4p(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  response = intensity,
  dose = concentration
)

## End(Not run)
```

`peptide_type` *Assign peptide type*

Description

Based on preceding and C-terminal amino acid, the peptide type of a given peptide is assigned. Peptides with preceeding and C-terminal lysine or arginine are considered fully-tryptic. If a peptide is located at the N- or C-terminus of a protein and fulfills the criterium to be fully-tryptic otherwise,

it is also considered as fully-tryptic. Peptides that only fulfill the criterium on one terminus are semi-tryptic peptides. Lastly, peptides that are not fulfilling the criteria for both termini are non-tryptic peptides.

Usage

```
peptide_type(
  data,
  aa_before = aa_before,
  last_aa = last_aa,
  aa_after = aa_after
)
```

Arguments

data	A data frame containing at least information about the preceding and C-terminal amino acids of peptides.
aa_before	The name of the column containing the preceding amino acid as one letter code.
last_aa	The name of the column containing the C-terminal amino acid as one letter code.
aa_after	The name of the column containing the following amino acid as one letter code.

Value

A data frame that contains the input data and an additional column with the peptide type information.

Examples

```
data <- data.frame(
  aa_before = c("K", "S", "T"),
  last_aa = c("R", "K", "Y"),
  aa_after = c("T", "R", "T")
)

peptide_type(data, aa_before, last_aa, aa_after)
```

---

plot_drc_4p *Plotting of four-parameter dose response curves*

Description

Function for plotting four-parameter dose response curves for each group (precursor, peptide or protein), based on output from `fit_drc_4p` function.

Usage

```
plot_drc_4p(
  data,
  grouping,
  response,
  dose,
  targets,
  unit = "uM",
  y_axis_name = "Response",
  facet = TRUE,
  scales = "free",
  x_axis_scale_log10 = TRUE,
  export = FALSE,
  export_name = "dose-response_curves"
)
```

Arguments

data	A data frame that is obtained by calling the `fit_drc_4p` function.
grouping	The name of the column containing precursor, peptide or protein identifiers.
response	The name of the column containing response values, eg. log2 transformed intensities.
dose	The name of the column containing dose values, eg. the treatment concentrations.
targets	A character vector that specifies the names of the precursors, peptides or proteins (depending on `grouping`) that should be plotted. This can also be `"all"` if plots for all curve fits should be created.
unit	A character vector specifying the unit of the concentration.
y_axis_name	A character vector specifying the name of the y-axis of the plot.
facet	A logical indicating if plots should be summarised into facets of 20 plots. This is recommended for many plots.
scales	A character vector that specifies if the scales in faceted plots (if more than one target was provided) should be `"free"` or `"fixed"`.
x_axis_scale_log10	A logical indicating if the x-axis scale should be log10 transformed.
export	A logical indicating if plots should be exported as PDF. The output directory will be the current working directory. The name of the file can be chosen using the `export_name` argument. If only one target is selected and `export = TRUE`, the plot is exported and in addition returned in R.
export_name	A character vector providing the name of the exported file if `export = TRUE`.

Value

If `targets = "all"` a list containing plots for every unique identifier in the `grouping` variable is created. Otherwise a plot for the specified targets is created with maximally 20 facets.

Examples

```
## Not run:
plot_drc_4p(
  data,
  grouping = eg_precursor_id,
  response = intensity,
  dose = concentration,
  targets = c("ABCDEFK")
)

## End(Not run)
```

`plot_peptide_profiles` *Peptide abundance profile plot*

Description

Creates a plot of peptide abundances across samples. This is helpful to investigate effects of peptide and protein abundance changes in different samples and conditions.

Usage

```
plot_peptide_profiles(
  data,
  sample,
  peptide,
  intensity_log2,
  grouping,
  targets,
  protein_abundance_plot = FALSE,
  interactive = FALSE,
  export = FALSE,
  export_name = "peptide_profile_plots"
)
```

Arguments

`data`	Data frame containing at least the input variables.
`sample`	Column in the data frame containing sample names.
`peptide`	Column in the data frame containing peptide or precursor names.
`intensity_log2`	Column in the data frame containing log2 transformed intensities.
`grouping`	Column in the data frame containing groups by which the data should be split. This can be for example protein IDs.
`targets`	Character vector specifying elements of the grouping column which should be plotted. This can also be `"all"` if plots for all groups should be created. Depending on the number of elements in your grouping column this can be many plots.

protein_abundance_plot
: Logical, if the input for this plot comes directly from calculate_protein_abundance this argument can be set to TRUE. This displays all peptides in gray, while the protein abundance is displayed in green.

interactive
: A logical indicating whether the plot should be interactive (default is FALSE). If this is TRUE only one target can be supplied to the function. Interactive plots cannot be exported either.

export
: A logical indicating if plots should be exported as PDF. The output directory will be the current working directory. The name of the file can be chosen using the export_name argument.

export_name
: A character vector providing the name of the exported file if export = TRUE.

Value

A list of peptide profile plots.

Examples

```
## Not run:
plot_peptide_abundance(
  data,
  sample = r_file_name,
  peptide = eg_precursor_id,
  intensity_log2 = log2_intensity,
  grouping = pg_protein_accessions,
  targets = c("P03421")
)

## End(Not run)
```

---

plot_pval_distribution

Plot histogram of p-value distribution

Description

Plots the distribution of p-values derived from any statistical test as a histogram.

Usage

```
plot_pval_distribution(data, grouping, pval)
```

Arguments

data
: a data frame containing at least grouping identifiers (precursor, peptide or protein) and p-values derived from any statistical test.

grouping
: the column in the data frame containing either precursor, peptide or protein identifiers.

pval
: the column in the data frame containing p-values.

Value

A histogram or boxplot that shows the intensity distribution over all samples or by sample.

Examples

```
## Not run:
plot_pval_distribution(
  data,
  grouping = eg_precursor_id,
  pval = pval
)

## End(Not run)
```

---

protti_colours *Colour scheme for protti*

Description

A colour scheme for protti that contains 100 colours.

Usage

```
protti_colours
```

Format

A vector containing 100 colours

Source

Dina's imagination.

---

qc_charge_states *Check charge state distribution*

Description

Calculates the charge state distribution for each sample (by count or intensity).

Usage

```
qc_charge_states(
  data,
  sample,
  grouping,
  charge_states,
  intensity = NULL,
  remove_na_intensities = TRUE,
  method = "count",
  plot = FALSE,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least sample names, peptide or precursor identifiers and missed cleavage counts for each peptide or precursor.
sample	the column in the data data frame containing the sample name.
grouping	the column in the data data frame containing either precursor or peptide identifiers.
charge_states	the column in the data data frame containing the different charge states assigned to the precursor or peptide.
intensity	the name of the column containing the corresponding raw or normalised intensity values (not log2) for each peptide or precursor. Required when "intensity" is chosen as the method.
remove_na_intensities	logical specifying if sample/grouping combinations with intensities that are NA (not quantified IDs) should be dropped from the data frame for analysis of missed cleavages. Default is TRUE since we are usually interested in quantifiable peptides. This is only relevant for method = "count".
method	character vector indicating the method used for evaluation. "count" calculates the charge state distribution based on counts of the corresponding peptides or precursors in the charge state group, "intensity" calculates the percentage of precursors or peptides in each charge state group based on the corresponding intensity values.
plot	logical indicating whether the result should be plotted.
interactive	argument specifying whether the plot should be interactive (default is FALSE).

Value

A data frame that contains the calculated percentage made up by the sum of either all counts or intensities of peptides or precursors of the corresponding charge state (depending on which method is chosen).

Examples

```
## Not run:
qc_charge_states(
  data,
  sample = r_file_name,
  grouping = pep_stripped_sequence,
  charge_states = fg_charge,
  method = "count",
  plot = TRUE
)

## End(Not run)
```

`qc_contaminants` *Percentage of contaminants per sample*

Description

Calculates the percentage of contaminating proteins as the share of total intensity

Usage

```
qc_contaminants(
  data,
  sample,
  protein,
  is_contaminant,
  intensity,
  n_contaminants = 5,
  plot = TRUE,
  interactive = FALSE
)
```

Arguments

`data`	a data frame containing at least the input variables.
`sample`	the name of the column containing the sample names.
`protein`	the name of the column containing protein IDs or protein names.
`is_contaminant`	the name of the column containing a logical indicating if the protein is a contaminant.
`intensity`	the name of the column containing the corresponding raw or untransformed normalised intensity values.
`n_contaminants`	numeric, indicating how many contaminants should be displayed individually. The rest is combined to a group called "other". The default is 5.
`plot`	logical, if TRUE a plot is returned. If FALSE a table is returned.
`interactive`	logical, if TRUE the plot is interactive using plotly.

Value

A bar plot that displays the percentage of contaminating proteins over all samples. If `plot = FALSE` a data frame is returned.

Examples

```
## Not run:
qc_contaminants(
  data,
  sample = sample,
  protein = leading_razor_protein,
  is_contaminant = potential_contaminant,
  intensity = intensity
)

## End(Not run)
```

`qc_cvs` *Check CV distribution*

Description

Calculates and plots the coefficients of variation for the selected grouping.

Usage

```
qc_cvs(
  data,
  grouping,
  condition,
  intensity,
  plot = TRUE,
  plot_style = "density"
)
```

Arguments

`data`	a data frame containing at least peptide, precursor or protein identifiers, information on conditions and intensity values for each peptide, precursor or protein.
`grouping`	the column in the input data frame containing the grouping variables (e.g. peptides, precursors or proteins).
`condition`	the column in the data data frame containing condition information (e.g. "treated" and "control").
`intensity`	the name of the column containing the corresponding raw or untransformed normalised intensity values for each peptide or precursor.
`plot`	logical indicating whether the result should be plotted. Default is TRUE.

plot_style — character vector indicating the plotting style. plot_style = "boxplot" plots a boxplot, whereas plot_style = "density" plots the CV density distribution. plot_style = "violin" returns a violin plot. Default is plot_style = "density".

Value

Either the median CVs in

Examples

```
## Not run:
qc_cvs(
  data,
  grouping = pep_stripped_sequence,
  condition = condition,
  intensity = fg_quantity,
  plot = TRUE,
  plot_style = "density"
)

## End(Not run)
```

---

qc_data_completeness *Data completeness*

Description

Calculates the percentage of data completeness. That means, what percentage of all detected precursors is present in each sample.

Usage

```
qc_data_completeness(
  data,
  sample,
  grouping,
  intensity,
  digestion = NULL,
  plot = TRUE,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least the input variables.
sample	The name of the column containing the sample names.
grouping	The name of the column containing either precursor or peptide identifiers.

intensity	The name of the column containing any intensity intensity values that missingness should be determined for.
digestion	Optional column indicating the mode of digestion (limited proteolysis or tryptic digest). Alternatively, any other variable by which the data should be split can be provided.
plot	Logical, if TRUE a plot is returned. If FALSE a table is returned.
interactive	Logical, if TRUE the plot is interactive using plotly.

Value

A bar plot that displays the percentage of data completeness over all samples. If `plot = FALSE` a data frame is returned. If `interactive = TRUE`, the plot is interactive.

Examples

```
## Not run:
qc_data_completeness(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity = fg_quantity,
  digestion = digestion
)

## End(Not run)
```

---

qc_ids *Check number of precursor, peptide or protein IDs*

Description

Returns a plot or table of the number of IDs for each sample. The default settings remove grouping variables without quantitative information (intensity is NA). These will not be counted as IDs.

Usage

```
qc_ids(
  data,
  sample,
  grouping,
  intensity,
  remove_na_intensities = TRUE,
  condition = NULL,
  title = "ID count per sample",
  plot = TRUE,
  interactive = FALSE
)
```

Arguments

data	a data frame containing at least sample names and precursor/peptide/protein IDs.
sample	the column in the data frame specifying the sample name.
grouping	the column in the data frame containing either precursor, peptide or protein identifiers.
intensity	the column in the data frame containing raw or log2 transformed intensities. If remove_na_intensities = FALSE, this argument is not required.
remove_na_intensities	logical specifying if sample/grouping combinations with intensities that are NA (not quantified IDs) should be dropped from the data frame. Default is TRUE since we are usually interested in the number of quantifiable IDs.
condition	optional column in the data frame specifying the condition of the sample (e.g. LiP_treated, LiP_untreated), if column is provided, the bars in the plot will be coloured according to the condition.
title	optional argument specifying the plot title (default is "ID count per sample").
plot	logical specifying whether the output of the function should be plotted (default is TRUE).
interactive	logical specifying whether the plot should be interactive (default is FALSE).

Value

A bar plot with the height corresponding to the number of IDs, each bar represents one sample (if plot = TRUE). If plot = FALSE a table with ID counts is returned.

Examples

```
## Not run:
qc_ids(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity = fg_quantity,
  condition = r_condition,
  title = "Number of peptide IDs per sample"
)

## End(Not run)
```

---

qc_intensity_distribution

Check intensity distribution per sample and overall

Description

Plots the overall or sample-wise distribution of all peptide intensities as a boxplot or histogram.

Usage

```
qc_intensity_distribution(
  data,
  sample = NULL,
  grouping,
  intensity_log2,
  plot_style
)
```

Arguments

data	A data frame containing at least sample names, grouping identifiers (precursor, peptide or protein) and log2 transformed intensities for each grouping identifier.
sample	The column in the data frame containing the sample name. NOTE: If the overall distribution should be returned please do not provide the name of the sample column.
grouping	The column in the data frame containing either precursor, peptide or protein identifiers.
intensity_log2	The column in the data frame containing the log2 transformed intensities of each grouping identifier sample combination.
plot_style	A character vector indicating the plot type. This can be either "histogram", "boxplot" or "violin". Plot style "boxplot" and "violin" can only be used if a sample column is provided.

Value

A histogram or boxplot that shows the intensity distribution over all samples or by sample.

Examples

```
## Not run:
qc_intensity_distribution(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity_log2 = normalised_intensity_log2,
  plot_style = "boxplot"
)

## End(Not run)
```

`qc_median_intensities` *Median run intensities*

Description

Median intensities per run are returned either as a plot or a table.

Usage

```
qc_median_intensities(
  data,
  sample,
  grouping,
  intensity,
  plot = TRUE,
  interactive = FALSE
)
```

Arguments

`data`	a data frame containing at least the input variables.
`sample`	the name of the column containing the sample names.
`grouping`	the name of the column containing precursor or peptide identifiers.
`intensity`	the name of the column containing intensity values. The intensity should be ideally log2 transformed, but also non-transformed values can be used.
`plot`	logical, if TRUE a plot is returned. If FALSE a table is returned.
`interactive`	logical, if TRUE the plot is interactive using plotly.

Value

A plot that displays median intensity over all samples. If `plot = FALSE` a data frame containing median intensities is returned.

Examples

```
## Not run:
qc_median_intensities(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity = log2_intensity
)

## End(Not run)
```

qc_missed_cleavages *Check missed cleavages*

Description

Calculates the percentage of missed cleavages for each sample (by count or intensity).The default settings remove grouping variables without quantitative information (intensity is NA). These will not be used for the calculation of missed cleavage percentages.

Usage

```
qc_missed_cleavages(
  data,
  sample,
  grouping,
  missed_cleavages,
  intensity,
  remove_na_intensities = TRUE,
  method = "count",
  plot = FALSE,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least sample names, peptide or precursor identifiers and missed cleavage counts for each peptide or precursor.
sample	the name of the column in the data data frame containing the sample name.
grouping	the name of the column in the data data frame containing either precursor or peptide identifiers.
missed_cleavages	the name of the column in the data data frame containing the counts of missed cleavages per peptide or precursor.
intensity	the name of the column containing the corresponding raw or normalised intensity values (not log2) for each peptide or precursor. Required when "intensity" is chosen as the method.
remove_na_intensities	Logical specifying if sample/grouping combinations with intensities that are NA (not quantified IDs) should be dropped from the data frame for analysis of missed cleavages. Default is TRUE since we are usually interested in quantifiable peptides. This is only relevant for method = "count".
method	character vector indicating the method used for evaluation. "count" calculates the percentage of missed cleavages based on counts of the corresponding peptide or precursor, "intensity" calculates the percentage of missed cleavages by intensity of the corresponding peptide or precursor.
plot	A logical indicating whether the result should be plotted.
interactive	Argument specifying whether the plot should be interactive (default is FALSE).

Value

A data frame that contains the calculated percentage made up by the sum of all peptides or precursors containing the corresponding amount of missed cleavages.

Examples

```
## Not run:
qc_missed_cleavages(
  data,
  sample = r_file_name,
  grouping = pep_stripped_sequence,
  missed_cleavages = pep_nr_of_missed_cleavages,
  intensity = fg_quantity,
  method = "intensity",
  plot = TRUE
)

## End(Not run)
```

---

qc_pca *Plot principal component analysis*

Description

Plots a principal component analysis based on peptide or precursor intensities.

Usage

```
qc_pca(
  data,
  sample,
  grouping,
  intensity,
  condition,
  components = c("PC1", "PC2"),
  digestion = NULL,
  plot_style = "pca"
)
```

Arguments

data	a data frame containing sample names, peptide or precursor identifiers, corresponding intensities and a condition column indicating e.g. the treatment.
sample	the column in the data data frame containing the sample name.
grouping	the column in the data data frame containing either precursor or peptide identifiers.

intensity	the column in the data data frame containing containing the corresponding intensity values for each peptide or precursor.
condition	the column in the data data frame indicating the treatment or condition for each sample.
components	character vector indicating the two components that should be displayed in the plot. By default these are PC1 and PC2. You can provide these using a character vector of the form c("PC1", "PC2").
digestion	optional column indicating the mode of digestion (limited proteolysis or tryptic digest).
plot_style	character vector specifying what plot should be returned. If ‘plot_style = "pca"‘ is selected the two PCA components supplied with the ‘components‘ argument are plottet against each other. This is the default. ‘plot_style = "scree"‘ returns a scree plot that displays the variance explained by each principal component in percent. The scree is useful for checking if any other than the default first two components should be plotted.

Value

A plotted principal component analysis showing PC1 and PC2

Examples

```
## Not run:
qc_pca(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity = normalised_intensity_log2,
  condition = r_condition,
  components = c("PC2", "PC3"),
  plot_style = "scree"
)

## End(Not run)
```

---

qc_peak_width *Peak width over retention time*

Description

Plots one minute binned median precursor elution peak width over retention time for each sample.

Usage

```
qc_peak_width(
  data,
  sample,
  intensity,
  retention_time,
  peak_width = NULL,
  retention_time_start = NULL,
  retention_time_end = NULL,
  remove_na_intensities = TRUE,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least sample names and protein IDs.
sample	The column in the data frame containing the sample names.
intensity	The column in the data frame containing intensities. If remove_na_intensities = FALSE, this argument is not required.
retention_time	The column in the data frame containing retention times of precursors.
peak_width	The column in the data frame containing peak width information. It is not required if retention_time_start and retention_time_end columns are provided.
retention_time_start	The column in the data frame containing the start time of the precursor elution peak. It is not required if the peak_width column is provided.
retention_time_end	The column in the data frame containing the end time of the precursor elution peak. It is not required if the peak_width column is provided.
remove_na_intensities	Logical specifying if sample/grouping combinations with intensities that are NA (not quantified IDs) should be dropped from the data frame. Default is TRUE since we are usually interested in the peak width of quantifiable data.
interactive	A logical indicating whether the plot should be interactive (default is FALSE).

Value

A line plot displaying one minute binned median precursor elution peak width over retention time for each sample.

Examples

```
## Not run:
qc_peak_width(
  data,
  sample = r_file_name,
  intensity = fg_quantity,
```

```
  retention_time = eg_mean_apex_rt,
  retention_time_start = eg_start_rt,
  retention_time_end = eg_end_rt
)

## End(Not run)
```

`qc_peptide_type` *Check peptide type percentage share*

Description

Calculates the percentage share of each peptide types (fully-tryptic, semi-tryptic, non-tryptic) for each sample.

Usage

```
qc_peptide_type(
  data,
  sample,
  peptide,
  pep_type,
  intensity,
  remove_na_intensities = TRUE,
  method = "count",
  plot = FALSE,
  interactive = FALSE
)
```

Arguments

`data`	A data frame containing at least the input columns.
`sample`	the name of the column containing the sample names.
`peptide`	the name of the column containing the peptide sequence.
`pep_type`	the name of the column containing the peptide type. Can be obtained using the `find_peptide` and `peptide_type` function together.
`intensity`	the name of the column containing the corresponding raw or normalised intensity values (not log2) for each peptide or precursor. Required when "intensity" is chosen as the method.
`remove_na_intensities`	Logical specifying if sample/peptide combinations with intensities that are NA (not quantified IDs) should be dropped from the data frame for analysis of peptide type distributions. Default is TRUE since we are usually interested in the peptide type distribution of quantifiable IDs. This is only relevant for method = "count".

method	character vector indicating the method used for evaluation. method = "intensity" calculates the peptide type percentage by intensity, whereas method = "count" calculates the percentage by peptide ID count. Default is method = count.
plot	a logical indicating whether the result should be plotted.
interactive	a logical indicating whether the plot should be interactive.

Value

A data frame that contains the calculated percentage shares of each peptide type per sample. The count column contains the number of peptides with a specific type. The peptide_type_percent column contains the percentage share of a specific peptide type.

Examples

```
## Not run:
qc_peptide_type(
  data,
  sample = r_file_name,
  peptide = pep_stripped_sequence,
  pep_type = pep_type,
  intensity = fg_quantity,
  method = "intensity",
  plot = TRUE
)

## End(Not run)
```

---

qc_proteome_coverage *Proteome coverage per sample and total*

Description

Calculates the proteome coverage for each samples and for all samples combined. In other words the fraction of detected proteins to all proteins in the proteome is calculated.

Usage

```
qc_proteome_coverage(
  data,
  sample,
  protein_id,
  organism_id,
  plot = TRUE,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least sample names and protein ID's.
sample	The column in the data data frame containing the sample name.
protein_id	The column in the data data frame containing protein identifiers such as UniProt accessions.
organism_id	The NCBI taxonomy identifier (TaxId) of the organism used. Human: 9606, S. cerevisiae: 559292, E. coli: 83333.
plot	A logical indicating whether the result should be plotted (default is TRUE).
interactive	A logical indicating whether the plot should be interactive (default is FALSE).

Value

A bar plot showing the percentage of of the proteome detected and undetected in total and for each sample. If `plot = FALSE` a data frame containing the numbers is returned.

Examples

```
## Not run:
qc_proteome_coverage(
  data,
  sample = r_file_name,
  protein_id = pg_protein_accession,
  organism_id = 9606
)

## End(Not run)
```

---

qc_sample_correlation *Correlation based hirachical clustering of samples*

Description

A correlation heatmap is created that uses hirachical clustering to determine sample similarity.

Usage

```
qc_sample_correlation(
  data,
  sample,
  grouping,
  intensity_log2,
  condition,
  digestion = NULL,
  run_order = NULL,
  method = "spearman",
  interactive = FALSE
)
```

Arguments

data	a dataframe contains at least the input variables.
sample	the name of the column containing the sample names.
grouping	the name of the column containing precursor or peptide identifiers.
intensity_log2	the name of the column containing log2 intensity values.
condition	the name of the column containing the conditions.
digestion	optional, the name of the column containing information about the digestion method used. Eg. "LiP" or "tryptic control".
run_order	optional, the name of the column containing the order in which samples were measured. Useful to investigate batch effects due to run order.
method	the method to be used for correlation. "spearman" is the default but can be changed to "pearson" or "kendall".
interactive	logical, default is FALSE. Determines if an interactive or static heatmap should be created using heatmaply or pheatmap, respectively.

Value

A correlation heatmap that compares each sample. The dendrogram is sorted by optimal leaf ordering.

Examples

```
## Not run:
qc_sample_correlation(
  data,
  sample = r_file_name,
  grouping = eg_precursor_id,
  intensity_log2 = intensity_log2,
  condition = r_condition
)

## End(Not run)
```

---

qc_sequence_coverage *Protein coverage distribution*

Description

Plots the distribution of protein coverages in a histogram.

Usage

```
qc_sequence_coverage(
  data,
  protein_identifier,
  coverage,
  sample = NULL,
  interactive = FALSE
)
```

Arguments

data	A data frame containing at least the input variables.
protein_identifier	Column in the data frame containing protein identifiers.
coverage	Column in the data frame containing protein coverage in percent. This information can be obtained using the sequence_coverage function.
sample	Column in the data frame containing sample names. Please only provide this argument if you want to facet the distribution plot by sample otherwise do not provide this argument.
interactive	A logical, if TRUE the plot is interactive (default is TRUE).

Value

A protein coverage histogram with 5 percent binwidth. The vertical dotted line indicates the median.

See Also

sequence_coverage

Examples

```
## Not run:
qc_sequence_coverage(
  data,
  protein_identifier = pg_protein_accessions,
  coverage = coverage
)

## End(Not run)
```

randomise_queue *Randomise samples in MS queue*

Description

This function randomises the order of samples in an MS queue. QC and Blank samples are left in place. It is also possible to randomise only parts of the queue. Before running this make sure to set a specific seed with the `set.seed()` function. This ensures that the randomisation of the result is consistent if the function is run again.

Usage

```
randomise_queue(data = NULL, rows = NULL, export = FALSE)
```

Arguments

`data`	optional, a data frame containing a queue. If not provided a queue file can be chosen interactively.
`rows`	optional, a range of rows in for which samples should be randomized.
`export`	logical, if TRUE a `"randomised_queue.csv"` file will be saved in the working directory. If FALSE a data frame will be returned.

Value

If `export = TRUE` a `"randomised_queue.csv"` file will be saved in the working directory. If `export = FALSE` a data frame that contains the randomised queue is returned.

Examples

```
## Not run:
randomise_queue(data = data, rows = 195:235, export = TRUE)

## End(Not run)
```

rapamycin_10uM *Rapamycin 10 uM example data*

Description

Rapamycin example data used for the vignette about binary control/treated data. The data is obtained from Piazza 2020 and corresponds to experiment 18. FKBP1A the rapamycin binding protein and 49 other randomly sampled proteins were used for this example dataset. Furthermore, only the DMSO control and the 10 uM condition were used.

Usage

```
rapamycin_10uM
```

Format

A data frame containing peptide level data from a Spectronaut report.

Source

Piazza, I., Beaton, N., Bruderer, R. et al. A machine learning-based chemoproteomic approach to identify drug targets and binding sites in complex proteomes. Nat Commun 11, 4200 (2020). https://doi.org/10.1038/s41467-020-18071-x

---

`rapamycin_dose_response`

Rapamycin dose response example data

Description

Rapamycin example data used for the vignette about dose response data. The data is obtained from Piazza 2020 and corresponds to experiment 18. FKBP1A the rapamycin binding protein and 39 other randomly sampled proteins were used for this example dataset. The concentration range includes the following points: 0 (DMSO control), 10 pM, 100 pM, 1 nM, 10 nM, 100 nM, 1 uM, 10 uM and 100 uM.

Usage

```
rapamycin_dose_response
```

Format

A data frame containing peptide level data from a Spectronaut report.

Source

Piazza, I., Beaton, N., Bruderer, R. et al. A machine learning-based chemoproteomic approach to identify drug targets and binding sites in complex proteomes. Nat Commun 11, 4200 (2020). https://doi.org/10.1038/s41467-020-18071-x

read_protti *Read, clean and convert*

Description

The function uses the very fast `fread` function form the `data.table` package. The column names of the resulting data table are made more r-friendly using `clean_names` from the `janitor` package. It replaces "." and " " with "_" and converts names to lower case which is also known as snake_case. In the end the data table is converted to a tibble.

Usage

```
read_protti(filename, ...)
```

Arguments

`filename`	the path to the file.
`...`	additional arguments for the fread function.

Value

A data frame (with class tibble) that contains the content of the specified file.

Examples

```
## Not run:
read_protti("folder\\filename")

## End(Not run)
```

replace_identified_by_x *Replace identified positions in protein sequence by "x"*

Description

Helper function for the calculation of sequence coverage, replaces identified positions with an "x" within the protein sequence.

Usage

```
replace_identified_by_x(sequence, positions_start, positions_end)
```

Arguments

sequence	A character vector that contains the protein sequence.
positions_start	A vector of start positions of the identified peptides.
positions_end	A vector of end positions of the identified peptides.

Value

A character vector that contains the modified protein sequence with each identified position replaced by "x".

---

scale_protti *Scaling a vector*

Description

scale_protti is used to scale a numeric vector either between 0 and 1 or around a centered value using the standard deviation.

Usage

```
scale_protti(x, method)
```

Arguments

x	a numeric vector
method	the method to be used for scaling. "01" scales the vector between 0 and 1. "center" scales the vector equal to base::scale around a center. This is done by subtracting the mean from every value and then deviding them by the standard deviation.

Value

A scaled numeric vector.

Examples

```
scale_protti(c(1, 2, 1, 4, 6, 8), method = "01")
```

sequence_coverage *Protein sequence coverage*

Description

Calculate sequence coverage for each identified protein.

Usage

```
sequence_coverage(data, protein_sequence, peptides)
```

Arguments

data	A dataframe containing at least the protein sequence and the identified peptides as columns.
protein_sequence	A column containing protein sequences, can be obtained by using the function `fetch_uniprot()`
peptides	A column containing the identified peptides.

Value

A new column containing the calculated sequence coverages for each identified protein

Examples

```
data <- data.frame(
  protein_sequence = c("abcdefghijklmnop", "abcdefghijklmnop"),
  pep_stripped_sequence = c("abc", "jklmn")
)

sequence_coverage(
  data,
  protein_sequence = protein_sequence,
  peptides = pep_stripped_sequence
)
```

split_metal_name *Convert metal names to search pattern*

Description

Converts a vector of metal names extracted from the `feature_metal_binding` column obtained with `fetch_uniprot` to a pattern that can be used to search for corresponding ChEBI IDs. This is used as a helper function for other functions.

Usage

```
split_metal_name(metal_names)
```

Arguments

metal_names	A character vector containing names of metals and metal containing molecules.

Value

A character vector with metal name search patterns.

---

treatment_enrichment *Check treatment enrichment*

Description

Check for an enrichment of proteins interacting with the treatment in significantly changing proteins as compared to all proteins.

Usage

```
treatment_enrichment(
  data,
  protein_id,
  is_significant,
  binds_treatment,
  treatment_name,
  plot = TRUE
)
```

Arguments

data	A dataframe contains at least the input variables.
protein_id	The name of the column containing the protein accession numbers.
is_significant	The name of the column containing a logical indicating if the corresponding protein has a significantly changing peptide. The input data frame may contain peptide level information with significance information. The function is able to extract protein level information from this.
binds_treatment	The name of the column containing a logical indicating if the corresponding protein binds to the treatment. This information can be obtained from different databases, e.g Uniprot.
treatment_name	A character vector of the treatment name. It will be included in the plot title.
plot	A logical indicating whether the result should be plotted or returned as a table.

Value

A bar plot displaying the percentage of all detect proteins and all significant proteins that bind to the treatment. A Fisher's exact test is performed to calculate the significance of the enrichment in significant proteins compared to all proteins. The result is reported as a p-value. If `plot = FALSE` a contingency table in long format is returned.

Examples

```
## Not run:
treatment_enrichment(
  data,
  protein_id = pg_protein_accessions,
  is_significant = significant,
  binds_treatment = binds_metals,
  treatment = "Metals"
)

## End(Not run)
```

`try_query` *Query from URL*

Description

Downloads data table from URL. If an error occurs during the query (for example due to no connection) the function waits 3 seconds and tries again. If no result could be obtained after the given number of tries a message indicating the problem is returned.

Usage

```
try_query(
  url,
  max_tries = 5,
  silent = TRUE,
  header = TRUE,
  sep = "tab-separated-values"
)
```

Arguments

`url`	a character vector of an URL to the website that contains the table that should be downloaded.
`max_tries`	a numeric vector specifying the number of times the function tries to download the data in case an error occurs.
`silent`	a logical, if TRUE no individual messages are printed after each try that failed.
`header`	a logical, indicates if the first row of the data frame contains variable names.
`sep`	a character vector, specifying the separator of the table at the target URL. Options are "tab-separated-values" or "csv". Default is "tab-separated-values".

Value

A data frame that contains the table from the url.

`ttest_protti` *Perform Welch's t-test*

Description

Performs a Welch's t-test and calculates p-values between two groups.

Usage

```
ttest_protti(mean1, mean2, sd1, sd2, n1, n2, log_values = TRUE)
```

Arguments

`mean1`	Vector containing the means of group1.
`mean2`	Vector containing the means of group2.
`sd1`	Vector containing the standard deviations of group1.
`sd2`	Vector containing the standard deviations of group2.
`n1`	Vector containing the number of replicates used for the calculation of each mean and standard deviation of group1.
`n2`	Vector containing the number of replicates used for the calculation of each mean and standard deviation of group2.
`log_values`	Logical indicating if values are log transformed. This determines how fold changes are calculated. Default is `log_values = TRUE`.

Value

A data frame that contains the calculated differences of means, standard error, t statistic and p-values.

Examples

```
ttest_protti(
  mean1 = 10,
  mean2 = 15.5,
  sd1 = 1,
  sd2 = 0.5,
  n1 = 3,
  n2 = 3
)
```

volcano_protti *Volcano plot*

Description

Plots a volcano plot for the given input.

Usage

```
volcano_protti(
  data,
  grouping,
  log2FC,
  significance,
  method,
  target_column = NULL,
  target = NULL,
  facet_by = NULL,
  title = "Volcano plot",
  x_axis_label = "log2(fold change)",
  y_axis_label = "-log10(q-value)",
  legend_label = "Target",
  log2FC_cutoff = 1,
  significance_cutoff = 0.01,
  interactive = FALSE
)
```

Arguments

data	a data frame containing at least the input variables.
grouping	the column in the data data frame containing either precursor or peptide identifiers.
log2FC	the column in the data frame containing the log2 transfromed fold changes between two conditions.
significance	the column containing the p-value or adjusted p-value for the corresponding fold changes. P-value is ideally adjusted using e.g. Benjamini-Hochberg correction.
method	character verctor with the method used for the plot. `method = "target"` highlights your protein, proteins or any other entities of interest (specified in the 'target' argument) in the volcano plot. `method = "significant"` highlights all significantly changing entities.
target_column	optional column required for `method = "target"`, can contain for example protein identifiers or a logical that marks certain proteins such as proteins that are known to interact with the treatment. Can also be provided if `method = "significant"` to label data points in an interactive plot.

target	optional character vector argument required for method = "target". It can contain one or more specific entities of the column provided in target_column. This can be for example a protein ID if target_column contains protein IDs or TRUE or FALSE for a logical column.
facet_by	optional argument specifying a column that contains information by which the data should be faceted into multiple plots.
title	optional argument specifying the title of the volcano plot. Default is "Volcano plot".
x_axis_label	optional argument specifying the x-axis label. Default is "log2(fold change)".
y_axis_label	optional argument specifying the y-axis label. Default is "-log10(q-value)".
legend_label	optional argument specifying the legend label. Default is "Target".
log2FC_cutoff	optional argument specifying the log2 transformed fold change cutoff used for assessing whether changes are significant. Default value is 1.
significance_cutoff	optional argument specifying the p-value cutoff used for assessing significance of changes. Default is 0.01.
interactive	logical, indicating whether the plot should be interactive or not. Default is interactive = FALSE.

Value

Depending on the method used a volcano plot with either highlighted targets (method = "target") or highlighted significant proteins (method = "significant") is returned.

Examples

```
## Not run:
volcano_protti(
  data,
  grouping = pep_stripped_sequence,
  log2FC = log2FC,
  significance = p_value,
  method = "target",
  target_column = uniprot_id,
  target = "Q9Y6K9",
  facet_by = comparison,
  title = "Finding Nemo",
  x_axis_label = "log2(fold change) treated vs untreated",
  y_axis_label = "-log10(p-value)",
  legend_label = "Target Protein",
  log2FC_cutoff = 2,
  significance_cutoff = 0.05,
  interactive = TRUE
)

## End(Not run)
```

`woods_plot` *Wood's plot*

Description

Creates a Wood's plot that plots log2 fold change of peptides or precursors along the protein sequence.

Usage

```
woods_plot(
  data,
  fold_change,
  start_position,
  end_position,
  protein_length,
  coverage = NULL,
  protein_id = NULL,
  facet = NULL,
  colouring = NULL,
  fold_change_cutoff = 1
)
```

Arguments

`data`	Data frame containing differential abundance, start and end peptide or precursor positions, protein length and optionally a variable based on which peptides or precursors should be coloured.
`fold_change`	Column in the data frame containing log2 fold changes.
`start_position`	Column in the data frame containing the start positions for each peptide or precursor.
`end_position`	Column in the data frame containing the end positions for each peptide or precursor.
`protein_length`	Column in the data frame containing the length of the protein.
`coverage`	Optional, column in the data frame containing coverage in percent. Will appear in the title of the barcode if provided.
`protein_id`	Optional argument, column in the data frame containing protein identifiers. Required if only one protein should be plotted and the data frame contains only information for this protein.
`facet`	Optional argument, column in the data frame containing information by which data should be faceted. This can be protein identifiers.
`colouring`	Optional argument, column in the data frame containing information by which peptide or precursors should be coloured.
`fold_change_cutoff`	Optional argument specifying the log2 fold change cutoff used for assessing whether changes are significant. The default value is 2.

Value

A Wood's plot is returned. Plotting peptide or precursor fold changes accross protein sequence.

Examples

```
## Not run:
woods_plot(test,
  fold_change = diff,
  start_position = start,
  end_position = end,
  protein_length = length,
  colouring = pep_type,
  facet = pg_protein_accessions
)

## End(Not run)
```

Index