Package 'protti'

July 1, 2022

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

Version 0.3.1

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, readr, lifecycle, httr, methods

RoxygenNote 7.2.0

Suggests testthat, covr, knitr, rmarkdown, shiny, r3dmol, proDA, limma, dendextend, pheatmap, heatmaply, furrr, future, parallel, seriation, drc, 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] (<https://orcid.org/0000-0003-2713-778X>), Dina Schuster [aut] (<https://orcid.org/0000-0001-6611-8237>), ETH Zurich [cph, fnd]

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

Repository CRAN

Date/Publication 2022-07-01 08:00:05 UTC

R topics documented:

analyse_functional_network . . . 3
anova_protti . . . 5
assign_missingness . . . 7
assign_peptide_type . . . 9
barcode_plot . . . 10
calculate_aa_scores . . . 11
calculate_diff_abundance . . . 13
calculate_go_enrichment . . . 16
calculate_imputation . . . 19
calculate_kegg_enrichment . . . 20
calculate_protein_abundance . . . 22
calculate_sequence_coverage . . . 24
calculate_treatment_enrichment . . . 25
create_queue . . . 26
create_structure_contact_map . . . 30
create_synthetic_data . . . 33
drc_4p . . . 35
drc_4p_plot . . . 36
extract_metal_binders . . . 38
fetch_alphafold_prediction . . . 40
fetch_chebi . . . 42
fetch_eco . . . 43
fetch_go . . . 44
fetch_kegg . . . 45
fetch_metal_pdb . . . 45
fetch_mobidb . . . 48
fetch_pdb . . . 48
fetch_pdb_structure . . . 49
fetch_uniprot . . . 51
fetch_uniprot_proteome . . . 52
filter_cv . . . 53
find_all_subs . . . 54
find_chebis . . . 55
find_peptide . . . 55
find_peptide_in_structure . . . 56
fit_drc_4p . . . 58
impute . . . 62
map_peptides_on_structure . . . 64
normalise . . . 67
parallel_create_structure_contact_map . . . 68
parallel_fit_drc_4p . . . 71
peptide_profile_plot . . . 74

protti_colours . . . 76
ptsi_pgk . . . 77
pval_distribution_plot . . . 78
qc_charge_states . . . 79
qc_contaminants . . . 80
qc_cvs . . . 82
qc_data_completeness . . . 83
qc_ids . . . 85
qc_intensity_distribution . . . 86
qc_median_intensities . . . 88
qc_missed_cleavages . . . 89
qc_pca . . . 91
qc_peak_width . . . 92
qc_peptide_type . . . 94
qc_proteome_coverage . . . 96
qc_sample_correlation . . . 97
qc_sequence_coverage . . . 99
randomise_queue . . . 100
rapamycin_10uM . . . 101
rapamycin_dose_response . . . 102
read_protti . . . 103
replace_identified_by_x . . . 103
scale_protti . . . 104
split_metal_name . . . 105
try_query . . . 105
ttest_protti . . . 106
viridis_colours . . . 107
volcano_plot . . . 107
woods_plot . . . 110

Index **113**

`analyse_functional_network`
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

```
analyse_functional_network(
  data,
  protein_id,
  string_id,
  organism_id,
  version = "11.5",
  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`	a character column in the `data` data frame that contains the protein accession numbers.
`string_id`	a character column in the `data` data frame that contains STRING database identifiers. These can be obtained from UniProt.
`organism_id`	a numeric value specifying an organism ID (NCBI taxon-ID). This can be obtained from here. H. sapiens: 9606, S. cerevisiae: 4932, E. coli: 511145.
`version`	a character value that specifies the version of STRINGdb to be used. Default is 11.5.
`score_threshold`	a numeric value specifying the interaction score that 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`	a logical column in the `data` data frame that indicates if the corresponding protein binds to the treatment. This information can be obtained from different databases, e.g UniProt.
`halo_color`	optional, character value with a color hex-code. This is the color of the halo of proteins that bind the treatment.
`plot`	a logical that indicates 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 data frame with interaction information is returned.

Examples

```
# Create example data
data <- data.frame(
  uniprot_id = c(
    "P0A7R1",
    "P02359",
    "P60624",
    "P0A7M2",
    "P0A7X3",
    "P0AGD3"
  ),
  database_string = c(
    "511145.b4203;",
    "511145.b3341;",
    "511145.b3309;",
    "511145.b3637;",
    "511145.b3230;",
    "511145.b1656;"
  ),
  is_known = c(
    TRUE,
    TRUE,
    TRUE,
    TRUE,
    TRUE,
    FALSE
  )
)

# Perform network analysis
network <- analyse_functional_network(
  data,
  protein_id = uniprot_id,
  string_id = database_string,
  organism_id = 511145,
  binds_treatment = is_known,
  plot = TRUE
)

network
```

---

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	a character column in the data data frame that contains precursor or peptide identifiers.
condition	a character or numeric column in the data data frame that contains the conditions.
mean_ratio	a numeric column in the data data frame that contains mean intensities or mean intensity ratios.
sd	a numeric column in the data data frame that contains the standard deviation corresponding to the mean.
n	a numeric column in the data data frame that contains 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

```
data <- data.frame(
  precursor = c("A", "A", "A", "B", "B", "B"),
  condition = c("C1", "C2", "C3", "C1", "C2", "C3"),
  mean = c(10, 12, 20, 11, 12, 8),
  sd = c(2, 1, 1.5, 1, 2, 4),
  n = c(4, 4, 4, 4, 4, 4)
)

anova_protti(
  data,
  grouping = precursor,
  condition = condition,
  mean = mean,
  sd = sd,
  n = n
)
```

`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 = "all",
  completeness_MAR = 0.7,
  completeness_MNAR = 0.2,
  retain_columns = NULL
)
```

Arguments

`data`	a data frame containing at least the input variables.
`sample`	a character column in the `data` data frame that contains the sample name.
`condition`	a character or numeric column in the `data` data frame that contains the conditions.
`grouping`	a character column in the `data` data frame that contains precursor or peptide identifiers.
`intensity`	a numeric column in the `data` data frame that contains intensity values.
`ref_condition`	a character vector providing the condition that is used as a reference for missingness determination. Instead of providing one reference condition, "all" can be supplied, which will create all pairwise condition pairs. By default `ref_condition = "all"`.
`completeness_MAR`	a numeric value that specifies 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`	a numeric value that specifies 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 that indicates columns that 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.
- NA: The comparison is not complete enough to fall into any other category. It will not be imputed if imputation is performed. For statistical significance testing these comparisons are filtered out after the test and prior to p-value adjustment. This can be prevented by setting `filter_NA_missingness = FALSE` in the `calculate_diff_abundance()` function.

The type of missingness has an influence on the way values are imputeted if imputation is performed subsequently using the `impute()` function. How each type of missingness is specifically imputed can be found in the function description. The type of missingness assigned to a comparison does not have any influence on the statistical test in the `calculate_diff_abundance()` function.

Examples

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 10,
  frac_change = 0.5,
  n_replicates = 4,
  n_conditions = 2,
  method = "effect_random",
  additional_metadata = FALSE
)

head(data, n = 24)

# Assign missingness information
data_missing <- assign_missingness(
  data,
  sample = sample,
```

```
  condition = condition,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  ref_condition = "all",
  retain_columns = c(protein)
)

head(data_missing, n = 24)
```

`assign_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

```
assign_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`	a character column in the `data` data frame that contains the preceding amino acid as one letter code.
`last_aa`	a character column in the `data` data frame that contains the C-terminal amino acid as one letter code.
`aa_after`	a character column in the `data` data frame that contains 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")
)

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

---

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

```
data <- data.frame(
  start = c(5, 40, 55, 130, 181, 195),
  end = c(11, 51, 60, 145, 187, 200),
  length = rep(200, 6),
  pg_protein_accessions = rep("Protein 1", 6),
  diff = c(1, 2, 5, 2, 1, 1),
  pval = c(0.1, 0.01, 0.01, 0.2, 0.2, 0.01)
)

barcode_plot(
  data,
  start_position = start,
  end_position = end,
  protein_length = length,
  facet = pg_protein_accessions,
  cutoffs = c(diff = 2, pval = 0.05)
)
```

---

calculate_aa_scores *Calculate scores for each amino acid position in a protein sequence*

Description

[Experimental] Calculate a score for each amino acid position in a protein sequence based on the product of the -log10(adjusted p-value) and the absolute log2(fold change) per peptide covering this amino acid. In detail, all the peptides are aligned along the sequence of the corresponding protein, and the average score per amino acid position is computed. In a limited proteolysis coupled to mass spectrometry (LiP-MS) experiment, the score allows to prioritize and narrow down structurally affected regions.

Usage

```
calculate_aa_scores(
  data,
  protein,
  diff = diff,
  adj_pval = adj_pval,
  start_position,
  end_position,
  retain_columns = NULL
)
```

Arguments

data	a data frame containing at least the input columns.
protein	a character column in the data frame containing the protein identifier or name.
diff	a numeric column in the data data frame containing the log2 fold change.
adj_pval	a numeric column in the data data frame containing the adjusted p-value.
start_position	a numeric column data in the data frame containing the start position of a peptide or precursor.
end_position	a numeric column in the data frame containing the end position of a peptide or precursor.
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 aggregated scores per amino acid position, enabling to draw fingerprints for each individual protein.

Author(s)

Patrick Stalder

Examples

```
data <- data.frame(
  pg_protein_accessions = c(rep("protein_1", 10)),
  diff = c(2, -3, 1, 2, 3, -3, 5, 1, -0.5, 2),
  adj_pval = c(0.001, 0.01, 0.2, 0.05, 0.002, 0.5, 0.4, 0.7, 0.001, 0.02),
  start = c(1, 3, 5, 10, 15, 25, 28, 30, 41, 51),
  end = c(6, 8, 10, 16, 23, 35, 35, 35, 48, 55)
)
calculate_aa_scores(
  data,
  protein = pg_protein_accessions,
```

```
  diff = diff,
  adj_pval = adj_pval,
  start_position = start,
  end_position = end
)
```

`calculate_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

```
calculate_diff_abundance(
  data,
  sample,
  condition,
  grouping,
  intensity_log2,
  missingness = missingness,
  comparison = comparison,
  mean = NULL,
  sd = NULL,
  n_samples = NULL,
  ref_condition = "all",
  filter_NA_missingness = TRUE,
  method = c("moderated_t-test", "t-test", "t-test_mean_sd", "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`	a character column in the `data` data frame that contains the sample name. Is not required if `method = "t-test_mean_sd"`.
`condition`	a character or numeric column in the `data` data frame that contains the conditions.
`grouping`	a character column in the `data` data frame that contains precursor or peptide identifiers.
`intensity_log2`	a numeric column in the `data` data frame that contains intensity values. The intensity values need to be log2 transformed. Is not required if `method = "t-test_mean_sd"`.

missingness
: a character column in the data data frame that contains missingness information. Can be obtained by calling assign_missingness(). Is not required if method = "t-test_mean_sd". The type of missingness assigned to a comparison does not have any influence on the statistical test. However, if filter_NA_missingness = TRUE then comparisons with missingness NA are filtered out prior to p-value adjustment.

comparison
: a character column in the data data frame that contains information of treatment/reference condition pairs. Can be obtained by calling assign_missingness. Comparisons need to be in the form condition1_vs_condition2, meaning two compared conditions are separated by "_vs_". This column determines for which condition pairs differential abundances are calculated. Is not required if method = "t-test_mean_sd", in that case please provide a reference condition with the ref_condition argument.

mean
: a numeric column in the data data frame that contains mean values for two conditions. Is only required if method = "t-test_mean_sd".

sd
: a numeric column in the data data frame that contains standard deviations for two conditions. Is only required if method = "t-test_mean_sd".

n_samples
: a numeric column in the data data frame that contains the number of samples per condition for two conditions. Is only required if method = "t-test_mean_sd".

ref_condition
: optional, character value providing the condition that is used as a reference for differential abundance calculation. Only required for method = "t-test_mean_sd". Instead of providing one reference condition, "all" can be supplied, which will create all pairwise condition pairs. By default ref_condition = "all".

filter_NA_missingness
: a logical value, default is TRUE. For all methods except "t-test_mean_sd" missingness information has to be provided. This information can be for example obtained by calling assign_missingness(). If a reference/treatment pair has too few samples to be considered robust based on user defined cutoffs, it is annotated with NA as missingness by the assign_missingness() function. If this argument is TRUE, these NA reference/treatment pairs are filtered out after the testing and prior to p-value adjustment.

method
: a character value, 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 inferred from the model. More information can be found in the proDA documentation. We do not recommend using the moderated_t-test or proDA method if the data was filtered for low CVs or imputation was performed. Default is method = "moderated_t-test".

p_adj_method
: a character value, 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": Columns labeled as control refer to the second condition of the comparison pairs. Treated refers to the first condition. `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` contain the 2.5% and 97.5% confidence interval borders for 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. `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% that this group is differentially abundant. A B-statistic of zero corresponds to a 50-50 chance that the group is differentially 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

```
set.seed(123) # Makes example reproducible

# Create synthetic data
data <- create_synthetic_data(
  n_proteins = 10,
  frac_change = 0.5,
  n_replicates = 4,
  n_conditions = 2,
```

```
  method = "effect_random",
  additional_metadata = FALSE
)

# Assign missingness information
data_missing <- assign_missingness(
  data,
  sample = sample,
  condition = condition,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  ref_condition = "all",
  retain_columns = c(protein, change_peptide)
)

# Calculate differential abundances
# Using "moderated_t-test" and "proDA" improves
# true positive recovery progressively
diff <- calculate_diff_abundance(
  data = data_missing,
  sample = sample,
  condition = condition,
  grouping = peptide,
  intensity_log2 = peptide_intensity_missing,
  missingness = missingness,
  comparison = comparison,
  method = "t-test",
  retain_columns = c(protein, change_peptide)
)

head(diff, n = 10)
```

`calculate_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

```
calculate_go_enrichment(
  data,
```

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

Arguments

data	a data frame that contains at least the input variables.
protein_id	a character column in the data data frame that contains the protein accession numbers.
is_significant	a logical column in the data data frame that indicates 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, a character column in the data data frame that contains 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, character value 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, character value specifying 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 not obtained with fetch_go 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 argument indicating whether the result should be plotted or returned as a table.
label	a logical argument indicating whether labels should be added to the plot. Default is TRUE.
plot_cutoff	a character value 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

```
# Load libraries
library(dplyr)
library(stringr)

# Create example data
# Contains artificial de-enrichment for ribosomes.
data <- fetch_uniprot_proteome(
  organism_id = 83333,
  columns = c(
    "id",
    "go(molecular function)"
  )
) %>%
  mutate(significant = c(
    rep(TRUE, 1000),
    rep(FALSE, n() - 1000)
  )) %>%
  mutate(significant = ifelse(
    str_detect(
      go_molecular_function,
      pattern = "ribosome"
    ),
    FALSE,
    significant
  ))

# Plot gene ontology enrichment
calculate_go_enrichment(
  data,
  protein_id = id,
  go_annotations_uniprot = go_molecular_function,
  is_significant = significant,
  plot = TRUE,
  plot_cutoff = "pval 0.01"
)

# Calculate gene ontology enrichment
go_enrichment <- calculate_go_enrichment(
  data,
  protein_id = id,
  go_annotations_uniprot = go_molecular_function,
  is_significant = significant,
  plot = FALSE,
```

```
)

head(go_enrichment, n = 10)
```

`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`	a numeric value specifying the minimal intensity value of the precursor/peptide. Is only required if `method = "ludovic"` and `missingness = "MNAR"`.
`noise`	a numeric value specifying a noise value for the precursor/peptide. Is only required if `method = "noise"` and `missingness = "MNAR"`.
`mean`	a numeric value specifying the mean intensity value of the condition with missing values for a given precursor/peptide. Is only required if `missingness = "MAR"`.
`sd`	a numeric value specifying the mean of the standard deviation of all conditions for a given precursor/peptide.
`missingness`	a character value specifying the missingness type of the data determines how values for imputation are sampled. This can be `"MAR"` or `"MNAR"`.
`method`	a character value specifying 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`	a logical value, 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 returned.

Value

A value sampled from a normal distribution with the input parameters. Method specifics are applied to input parameters prior to sampling.

`calculate_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

```
calculate_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`	a character column in the `data` data frame that contains the protein accession numbers.
`is_significant`	a logical column in the `data` data frame that indicates 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`	a character column in the `data` data frame that contains KEGG pathway identifiers. These can be obtained from KEGG using `fetch_kegg`.
`pathway_name`	a character column in the `data` data frame that contains KEGG pathway names. These can be obtained from KEGG using `fetch_kegg`.
`plot`	a logical value indicating whether the result should be plotted or returned as a table.
`plot_cutoff`	a character value 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

```
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible

# Create example data
kegg_data <- fetch_kegg(species = "eco")

if(!is.null(kegg_data)){ # only proceed if information was retrieved
data <- kegg_data %>%
  group_by(uniprot_id) %>%
  mutate(significant = rep(sample(
    x = c(TRUE, FALSE),
    size = 1,
    replace = TRUE,
    prob = c(0.2, 0.8)
  ),
  n = n()
  ))

# Plot KEGG enrichment
calculate_kegg_enrichment(
  data,
  protein_id = uniprot_id,
  is_significant = significant,
  pathway_id = pathway_id,
  pathway_name = pathway_name,
  plot = TRUE,
  plot_cutoff = "pval 0.05"
)

# Calculate KEGG enrichment
kegg <- calculate_kegg_enrichment(
  data,
  protein_id = uniprot_id,
  is_significant = significant,
  pathway_id = pathway_id,
  pathway_name = pathway_name,
  plot = FALSE
)

head(kegg, n = 10)
}
```

`calculate_protein_abundance` *Label-free protein quantification*

Description

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

Usage

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

Arguments

`data`	a data frame that contains at least the input variables.
`sample`	a character column in the `data` data frame that contains the sample name.
`protein_id`	a character column in the `data` data frame that contains the protein accession numbers.
`precursor`	a character column in the `data` data frame that contains precursors.
`peptide`	a character column in the `data` data frame that contains 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`	a numeric column in the `data` data frame that contains log2 transformed precursor intensities.
`method`	a character value 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.
`for_plot`	a logical value 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 in a data frame. The later output is ideal for plotting with `qc_protein_abundance` and can be filtered to only include protein abundances.

Examples

```
# Create example data
data <- data.frame(
  sample = c(
    rep("S1", 6),
    rep("S2", 6),
    rep("S1", 2),
    rep("S2", 2)
  ),
  protein_id = c(
    rep("P1", 12),
    rep("P2", 4)
  ),
  precursor = c(
    rep(c("A1", "A2", "B1", "B2", "C1", "D1"), 2),
    rep(c("E1", "F1"), 2)
  ),
  peptide = c(
    rep(c("A", "A", "B", "B", "C", "D"), 2),
    rep(c("E", "F"), 2)
  ),
  intensity = c(
    rnorm(n = 6, mean = 15, sd = 2),
    rnorm(n = 6, mean = 21, sd = 1),
    rnorm(n = 2, mean = 15, sd = 1),
    rnorm(n = 2, mean = 15, sd = 2)
  )
)

data

# Calculate protein abundances
protein_abundance <- calculate_protein_abundance(
  data,
  sample = sample,
  protein_id = protein_id,
  precursor = precursor,
  peptide = peptide,
  intensity_log2 = intensity,
  method = "sum",
  for_plot = FALSE
)

protein_abundance
```

```
# Calculate protein abundances and retain precursor
# abundances that can be used in a peptide profile plot
complete_abundances <- calculate_protein_abundance(
  data,
  sample = sample,
  protein_id = protein_id,
  precursor = precursor,
  peptide = peptide,
  intensity_log2 = intensity,
  method = "sum",
  for_plot = TRUE
)

complete_abundances
```

`calculate_sequence_coverage` *Protein sequence coverage*

Description

Calculate sequence coverage for each identified protein.

Usage

```
calculate_sequence_coverage(data, protein_sequence, peptides)
```

Arguments

`data`	a data frame containing at least the protein sequence and the identified peptides as columns.
`protein_sequence`	a character column in the `data` data frame that contains protein sequences. Can be obtained by using the function `fetch_uniprot()`
`peptides`	a character column in the `data` data frame that contains the identified peptides.

Value

A new column in the `data` data frame containing the calculated sequence coverage for each identified protein

Examples

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

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

`calculate_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

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

Arguments

`data`	a data frame contains at least the input variables.
`protein_id`	a character column in the `data` data frame that contains the protein accession numbers.
`is_significant`	a logical column in the `data` data frame that indicates 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`	a logical column in the `data` data frame that indicates if the corresponding protein binds to the treatment. This information can be obtained from different databases, e.g. UniProt.
`treatment_name`	a character value that indicates the treatment name. It will be included in the plot title.
`plot`	a logical value 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

```
# Create example data
data <- data.frame(
  protein_id = c(paste0("protein", 1:50)),
  significant = c(
    rep(TRUE, 20),
    rep(FALSE, 30)
  ),
  binds_treatment = c(
    rep(TRUE, 10),
    rep(FALSE, 10),
    rep(TRUE, 5),
    rep(FALSE, 25)
  )
)

# Plot treatment enrichment
calculate_treatment_enrichment(
  data,
  protein_id = protein_id,
  is_significant = significant,
  binds_treatment = binds_treatment,
  treatment = "Rapamycin",
  plot = TRUE
)

# Calculate treatment enrichment
enrichment <- calculate_treatment_enrichment(
  data,
  protein_id = protein_id,
  is_significant = significant,
  binds_treatment = binds_treatment,
  plot = FALSE
)

enrichment
```

`create_queue` *Creates a mass spectrometer queue for Xcalibur*

Description

[Experimental] 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, character value indicating the start date of the measurements.
`instrument`	optional, character value indicating the instrument initials.
`user`	optional, character value indicating the user name.
`measurement_type`	optional, character value indicating the measurement type of the samples (e.g "DIA", "DDA", "library" etc.).
`experiment_name`	optional, character value indicating the name of the experiment.

digestion optional, character vector indicating the digestion types used in this experiment (e.g "LiP" and/or "tryptic control").

treatment_type_1
optional, character vector indicating the name of the treatment.

treatment_type_2
optional, character vector indicating the name of a second treatment that was combined with the first treatment.

treatment_dose_1
optional, numeric vector indicating the doses used for treatment 1. These can be concentrations or times etc.

treatment_dose_2
optional, numeric vector indicating the doses used for treatment 2. These can be concentrations or times etc.

treatment_unit_1
optional, character vector indicating the unit of the doses for treatment 1 (e.g min, mM, etc.).

treatment_unit_2
optional, character vector indicating the unit of the doses for treatment 2 (e.g min, mM, etc.).

n_replicates optional, a numeric value indicating the number of replicates used per sample.

number_runs a logical that specifies if file names should be numbered from 1:n instead of adding experiment information. Default is FALSE.

organism optional, character value indicating the name of the organism used.

exclude_combinations
optional, list of lists that contains vectors of treatment types and treatment doses of which combinations should be excluded from the final queue.

inj_vol a numeric value indicating 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 a character value indicating 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 a character value indicating 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 a character vector that contains 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
a character vector that contains 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, numeric value that specifies in which intervals a blank sample should be inserted.

`blank_position` a character value that specifies the plate position of the blank. Will be `NA` if not specified, but needs to be specified later on then.

`blank_method_path` a character value that specifies 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` a numeric value that specifies the injection volume of the blank sample. Will be `NA` if not specified, but needs to be specified later on then.

`export` a logical value that specifies if the 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` a logical value that specifies if the resulting queue should be appended to an already existing queue. If false result will be saved as `queue.csv`.

`queue_path` optional, a character value that specifies the 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\\DIA_120min",
  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_structure_contact_map` *Creates a contact map of all atoms from a structure file*

Description

Creates a contact map of a subset or of all atom or residue distances in a structure or AlphaFold prediction file. Contact maps are a useful tool for the identification of protein regions that are in close proximity in the folded protein. Additionally, regions that are interacting closely with a small molecule or metal ion can be easily identified without the need to open the structure in programs such as PyMOL or ChimeraX. For large datasets (more than 40 contact maps) it is recommended to use the `parallel_create_structure_contact_map()` function instead, regardless of if maps should be created in parallel or sequential.

Usage

```
create_structure_contact_map(
  data,
  data2 = NULL,
  id,
  chain = NULL,
  auth_seq_id = NULL,
  distance_cutoff = 10,
  pdb_model_number_selection = c(0, 1),
  return_min_residue_distance = TRUE,
  show_progress = TRUE,
  export = FALSE,
  export_location = NULL,
  structure_file = NULL
)
```

Arguments

`data`	a data frame containing at least a column with PDB ID information of which the name can be provided to the `id` argument. If only this column is provided, all atom or residue distances are calculated. Additionally, a chain column can be present in the data frame of which the name can be provided to the `chain` argument. If chains are provided, only distances of this chain relative to the rest of the structure are calculated. Multiple chains can be provided in multiple rows. If chains are provided for one structure but not for another, the rows should contain NAs. Furthermore, specific residue positions can be provided in the `auth_seq_id` column if the selection should be further reduced. It is not

recommended to create full contact maps for more than a few structures due to time and memory limitations. If contact maps are created only for small regions it is possible to create multiple maps at once. By default distances of regions provided in this data frame to the complete structure are computed. If distances of regions from this data frame to another specific subset of regions should be computed, the second subset of regions can be provided through the optional `data2` argument.

`data2` optional, a data frame that contains a subset of regions for which distances to regions provided in the `data` data frame should be computed. If regions from the `data` data frame should be compared to the whole structure, data2 does not need to be provided. This data frame should have the same structure and column names as the `data` data frame.

`id` a character column in the `data` data frame that contains PDB or UniProt IDs for structures or AlphaFold predictions of which contact maps should be created. If a structure not downloaded directly from PDB is provided (i.e. a locally stored structure file) to the `structure_file` argument, this column should contain "my_structure" as content.

`chain` optional, a character column in the `data` data frame that contains chain identifiers for the structure file. Identifiers defined by the structure author should be used. Distances will be only calculated between the provided chains and the rest of the structure.

`auth_seq_id` optional, a character (or numeric) column in the `data` data frame that contains semicolon separated positions of regions for which distances should be calculated. This always needs to be provided in combination with a corresponding chain in `chain`. The position should match the positioning defined by the structure author. For PDB structures this information can be obtained from the `find_peptide_in_structure` function. The corresponding column in the output is called `auth_seq_id`. If an AlphaFold prediction is provided, UniProt positions should be used. If signal positions and not stretches of amino acids are provided, the column can be numeric and does not need to contain the semicolon separator.

`distance_cutoff`
a numeric value specifying the distance cutoff in Angstrom. All values for pairwise comparisons are calculated but only values smaller than this cutoff will be returned in the output. If a cutoff of e.g. 5 is selected then only residues with a distance of 5 Angstrom and less are returned. Using a small value can reduce the size of the contact map drastically and is therefore recommended. The default value is 10.

`pdb_model_number_selection`
a numeric vector specifying which models from the structure files should be considered for contact maps. E.g. NMR models often have many models in one file. The default for this argument is c(0, 1). This means the first model of each structure file is selected for contact map calculations. For AlphaFold predictions the model number is 0 (only .pdb files), therefore this case is also included here.

`return_min_residue_distance`
a logical value that specifies if the contact map should be returned for all atom distances or the minimum residue distances. Minimum residue distances are

smaller in size. If atom distances are not strictly needed it is recommended to set this argument to TRUE. The default is TRUE.

`show_progress` a logical value that specifies if a progress bar will be shown (default is TRUE).

`export` a logical value that indicates if contact maps should be exported as ".csv". The name of the file will be the structure ID. Default is `export = FALSE`.

`export_location`
optional, a character value that specifies the path to the location in which the contact map should be saved if `export = TRUE`. If left empty, they will be saved in the current working directory. The location should be provided in the following format "folderA/folderB".

`structure_file` optional, a character value that specifies the path to the location and name of a structure file in ".cif" or ".pdb" format for which a contact map should be created. All other arguments can be provided as usual with the exception of the `id` column in the `data` data frame, which should not contain a PDB or UniProt ID but a character vector containing only "my_structure".

Value

A list of contact maps for each PDB or UniProt ID provided in the input is returned. If the `export` argument is TRUE, each contact map will be saved as a ".csv" file in the current working directory or the location provided to the `export_location` argument.

Examples

```
# Create example data
data <- data.frame(
  pdb_id = c("6NPF", "1C14", "3NIR"),
  chain = c("A", "A", NA),
  auth_seq_id = c("1;2;3;4;5;6;7", NA, NA)
)

# Create contact map
contact_maps <- create_structure_contact_map(
  data = data,
  id = pdb_id,
  chain = chain,
  auth_seq_id = auth_seq_id,
  return_min_residue_distance = TRUE
)

str(contact_maps[["3NIR"]])

contact_maps
```

`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 = "effect_random",
  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`	a numeric value that specifies the number of proteins in the synthetic dataset.
`frac_change`	a numeric value that specifies the fraction of proteins that has a peptide changing in abundance. So far only one peptide per protein is changing.
`n_replicates`	a numeric value that specifies the number of replicates per condition.
`n_conditions`	a numeric value that specifies the number of conditions.
`method`	a character value that specifies the method type for the random sampling of significantly changing peptides. If `method = "effect_random"`, 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` a numeric vector of length equal to the 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`
: a numeric value that specifies the standard deviation of normal distribution that is used for sampling of inter-sample-differences. Default is 0.05.

`mean_protein_intensity`
: a numeric value that specifies the mean of the protein intensity distribution. Default: 16.8.

`sd_protein_intensity`
: a numeric value that specifies the standard deviation of the protein intensity distribution. Default: 1.4.

`mean_n_peptides`
: a numeric value that specifies the mean number of peptides per protein. Default: 12.75.

`size_n_peptides`
: a numeric value that specifies the 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`
: a numeric value that specifies the mean of peptide intensity standard deviations within a protein. Default: 1.7.

`sd_sd_peptides` a numeric value that specifies the standard deviation of peptide intensity standard deviation within a protein. Default: 0.75.

`mean_log_replicates, sd_log_replicates`
: a numeric value that specifies the `meanlog` and `sdlog` 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` a numeric value that specifies the 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`
: a numeric value that specifies the 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`
: a numeric value that specifies the standard deviation of the probabilistic dropout curve. Needs to be negative to sample a droupout towards low intensities. Default: -1.2.

`additional_metadata`
: a logical value that 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
```

---

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	a numeric column that contains the response values.
dose	a numeric column that contains the dose values.
log_logarithmic	a logical value 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".

---

drc_4p_plot *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

```
drc_4p_plot(
  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	a character column in the data data frame that contains the precursor, peptide or protein identifiers.
response	a numeric column in a nested data frame called plot_points that is part of the data data frame. This column contains the response values, e.g. log2 transformed intensities.
dose	a numeric column in a nested data frame called plot_points that is part of the data data frame. This column contains the dose values, e.g. 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 value specifying the unit of the concentration.
y_axis_name	a character value specifying the name of the y-axis of the plot.
facet	a logical value that indicates if plots should be summarised into facets of 20 plots. This is recommended for many plots.
scales	a character value 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 value that indicates if the x-axis scale should be log10 transformed.
export	a logical value that indicates 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 value 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 2,
  frac_change = 1,
  n_replicates = 3,
  n_conditions = 8,
  method = "dose_response",
  concentrations = c(0, 1, 10, 50, 100, 500, 1000, 5000),
  additional_metadata = FALSE
```

```
)

# Perform dose response curve fit
drc_fit <- fit_drc_4p(
  data = data,
  sample = sample,
  grouping = peptide,
  response = peptide_intensity_missing,
  dose = concentration,
  retain_columns = c(protein)
)

str(drc_fit)

# Plot dose response curves
if (!is.null(drc_fit)) {
  drc_4p_plot(
    data = drc_fit,
    grouping = peptide,
    response = peptide_intensity_missing,
    dose = concentration,
    targets = c("peptide_2_1", "peptide_2_3"),
    unit = "pM"
  )
}
```

`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	a character column in the data data frame that contains the protein identifiers.
feature_metal_binding	a character column in the data data frame that contains the feature metal binding information from UniProt.
chebi_cofactor	a character column in the data data frame that contains the ChEBI cofactor information from UniProt.
chebi_catalytic_activity	a character column in the data data frame that contains the 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

```
# Create example data
data <- fetch_uniprot(
  uniprot_ids = c("Q03640", "Q03778", "P22276"),
  columns = c(
    "feature(METAL BINDING)",
    "chebi(Cofactor)",
    "chebi(Catalytic activity)"
  )
)

# Extract metal binding information
metal_info <- extract_metal_binders(
  data = data,
  protein_id = id,
  feature_metal_binding = feature_metal_binding,
  chebi_cofactor = chebi_cofactor,
  chebi_catalytic_activity = chebi_catalytic_activity
)

metal_info
```

---

fetch_alphafold_prediction

Fetch AlphaFold prediction

Description

Fetches atom level data for AlphaFold predictions either for selected proteins or whole organisms.

Usage

```
fetch_alphafold_prediction(
  uniprot_ids = NULL,
  organism_name = NULL,
  timeout = 3600,
  return_data_frame = FALSE,
  show_progress = TRUE
)
```

Arguments

uniprot_ids	optional, a character vector of UniProt identifiers for which predictions should be fetched. This argument is mutually exclusive to the organism_name argument.

`organism_name` optional, a character value providing the name of an organism for which all available AlphaFold predictions should be retreived. The name should be the capitalised scientific species name (e.g. "Homo sapiens"). **Note:** Some organisms contain a lot of predictions which might take a considerable amount of time and memory to fetch. Therefore, you should be sure that your system can handle fetching predictions for these organisms. This argument is mutually exclusive to the `uniprot_ids` argument.

`timeout` a numeric value specifying the time in seconds until the download of an organism archive times out. The default is 3600 seconds.

`return_data_frame` a logical value that specifies if true, a data frame instead of a list is returned. It is recommended to only use this if not many pdb structures are retrieved. Default is FALSE.

`show_progress` a logical value that specifies if true, a progress bar will be shown. Default is TRUE.

Value

A list that contains atom level data for AlphaFold predictions. If return_data_frame is TRUE, a data frame with this information is returned instead. The data frame contains the following columns:

- label_id: Uniquely identifies every atom in the prediction following the standardised convention for mmCIF files.
- type_symbol: The code used to identify the atom species representing this atom type. This code is the element symbol.
- label_atom_id: Uniquely identifies every atom for the given residue following the standardised convention for mmCIF files.
- label_comp_id: A chemical identifier for the residue. This is the three- letter code for the amino acid.
- label_asym_id: Chain identifier following the standardised convention for mmCIF files. Since every prediction only contains one protein this is always "A".
- label_seq_id: Uniquely and sequentially identifies residues for each protein. The numbering corresponds to the UniProt amino acid positions.
- x: The x coordinate of the atom.
- y: The y coordinate of the atom.
- z: The z coordinate of the atom.
- prediction_score: Contains the prediction score for each residue.
- auth_seq_id: Same as `label_seq_id`. But of type character.
- auth_comp_id: Same as `label_comp_id`.
- auth_asym_id: Same as `label_asym_id`.
- uniprot_id: The UniProt identifier of the predicted protein.
- score_quality: Score annotations.

Examples

```
alphafold <- fetch_alphafold_prediction(
  uniprot_ids = c("F4HVG8", "O15552"),
  return_data_frame = TRUE
)

head(alphafold, n = 10)
```

`fetch_chebi` *Fetch ChEBI database information*

Description

Fetches all information from the ChEBI database.

Usage

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

Arguments

`relation`	a logical value that indicates if 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 each other. Default is FALSE.

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

```
chebi <- fetch_chebi()

head(chebi)
```

`fetch_eco` *Fetch evidence & conclusion ontology*

Description

Fetches all evidence & conclusion ontology (ECO) information from the EBI database. The ECO project is maintained through a public GitHub repository.

Usage

```
fetch_eco(
  return_relation = FALSE,
  return_history = FALSE,
  show_progress = TRUE
)
```

Arguments

`return_relation`	a logical value that indicates if relational information should be returned instead the main descriptive information. This data can be used to check the relations of ECO terms to each other. Default is FALSE.
`return_history`	a logical value that indicates if the entry history of an ECO term should be returned instead the main descriptive information. Default is FALSE.
`show_progress`	a logical value that indicates if a progress bar will be shown. Default is TRUE.

Details

According to the GitHub repository ECO is defined as follows:

"The Evidence & Conclusion Ontology (ECO) describes types of scientific evidence within the biological research domain that arise from laboratory experiments, computational methods, literature curation, or other means. Researchers use evidence to support conclusions that arise out of scientific research. Documenting evidence during scientific research is essential, because evidence gives us a sense of why we believe what we think we know. Conclusions are asserted as statements about things that are believed to be true, for example that a protein has a particular function (i.e. a protein functional annotation) or that a disease is associated with a particular gene variant (i.e. a phenotype-gene association). A systematic and structured (i.e. ontological) classification of evidence allows us to store, retreive, share, and compare data associated with that evidence using computers, which are essential to navigating the ever-growing (in size and complexity) corpus of scientific information."

More information can be found in their publication.

Value

A data frame that contains descriptive information about each ECO term in the EBI database. If either `return_relation` or `return_history` is set to `TRUE`, the respective information is returned instead of the usual output.

Examples

```
eco <- fetch_eco()

head(eco)
```

`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`	a character value 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

```
go <- fetch_go("9606")

head(go)
```

`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 value 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

```
kegg <- fetch_kegg(species = "hsa")

head(kegg)
```

`fetch_metal_pdb` *Fetch structural information about protein-metal binding from MetalPDB*

Description

Fetches information about protein-metal binding sites from the MetalPDB database. A complete list of different possible search queries can be found on their website.

Usage

```
fetch_metal_pdb(
  id_type = "uniprot",
  id_value,
  site_type = NULL,
  pfam = NULL,
  cath = NULL,
  scop = NULL,
  representative = NULL,
  metal = NULL,
  ligands = NULL,
  geometry = NULL,
  coordination = NULL,
  donors = NULL,
  columns = NULL,
  show_progress = TRUE
)
```

Arguments

id_type	a character value that specifies the type of the IDs provided to id_value. Default is "uniprot". Possible options include: "uniprot", "pdb", "ec_number", "molecule" and "organism".
id_value	a character vector supplying IDs that are of the ID type that was specified in id_type. E.g. UniProt IDs. Information for these IDs will be retreived.
site_type	optional, a character value that specifies a nuclearity for which information should be retrieved. The specific nuclearity can be supplied as e.g. "tetranuclear".
pfam	optional, a character value that specifies a Pfam domain for which information should be retrieved. The domain can be specified as e.g. "Carb_anhydrase".
cath	optional, a character value that specifies a CATH ID for which information should be retrieved. The ID can be specified as e.g. "3.10.200.10".
scop	optional, a character value that specifies a SCOP ID for which information should be retrieved. The ID can be specified as e.g. "b.74.1.1".
representative	optional, a logical that indicates if only information of representative sites of a family should be retrieved it can be specified here. A representative site is a site selected to represent a cluster of equivalent sites. The selection is done by choosing the PDB structure with the best X-ray resolution among those containing the sites in the cluster. NMR structures are generally discarded in favor of X-ray structures, unless all the sites in the cluster are found in NMR structures. If it is TRUE, only representative sites are retrieved, if it is FALSE, all sites are retrieved.
metal	optional, a character value that specifies a metal for which information should be retrieved. The metal can be specified as e.g. "Zn".
ligands	optional, a character value that specifies a metal ligand residue for which information should be retrieved. The ligand can be specified as e.g. "His".

geometry	optional, a character value that specifies a metal site geometry for which information should be retrieved. The geometry can be specified here based on the three letter code for geometries provided on their website.
coordination	optional, a character value that specifies a coordination number for which information should be retrieved. The number can be specified as e.g. "3".
donors	optional, a character value that specifies a metal ligand atom for which information should be retrieved. The atom can be specified as e.g. "S" for sulfur.
columns	optional, a character vector that specifies specific columns that should be retrieved based on the MetalPDB website. If nothing is supplied here, all possible columns will be retrieved.
show_progress	logical, if true, a progress bar will be shown. Default is TRUE.

Value

A data frame that contains information about protein-metal binding sites. The data frame contains some columns that might not be self explanatory.

- auth_id_metal: Unique structure atom identifier of the metal, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- auth_seq_id_metal: Residue identifier of the metal, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- pattern: Metal pattern for each metal bound by the structure.
- is_representative: A representative site is a site selected to represent a cluster of equivalent sites. The selection is done by choosing the PDB structure with the best X-ray resolution among those containing the sites in the cluster. NMR structures are generally discarded in favor of X-ray structures, unless all the sites in the cluster are found in NMR structures.
- auth_asym_id_ligand: Chain identifier of the metal-coordinating ligand residues, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- auth_seq_id_ligand: Residue identifier of the metal-coordinating ligand residues, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- auth_id_ligand: Unique structure atom identifier of the metal-coordinating ligand r esidues, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- auth_atom_id_ligand: Unique residue specific atom identifier of the metal-coordinating ligand residues, which is provided by the author of the structure in order to match the identification used in the publication that describes the structure.

Examples

```
head(fetch_metal_pdb(id_value = c("P42345", "P00918")))

fetch_metal_pdb(id_type = "pdb", id_value = c("1g54"), metal = "Zn")
```

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	a character value that specifies the 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("P0A799", "P62707")
)
```

fetch_pdb *Fetch structure information from RCSB*

Description

Fetches structure metadata from RCSB. If you want to retrieve atom data such as positions, use the function fetch_pdb_structure().

Usage

```
fetch_pdb(pdb_ids, batchsize = 200, show_progress = TRUE)
```

Arguments

pdb_ids	a character vector of PDB identifiers.
batchsize	a numeric value that specifies the number of structures to be processed in a single query. Default is 2000.
show_progress	a logical value that indicates if a progress bar will be shown. Default is TRUE.

Value

A data frame that contains structure metadata for the PDB IDs provided. The data frame contains some columns that might not be self explanatory.

- auth_asym_id: Chain identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- label_asym_id: Chain identifier following the standardised convention for mmCIF files.
- entity_beg_seq_id, ref_beg_seq_id, length, pdb_sequence: `entity_beg_seq_id` is a position in the structure sequence (`pdb_sequence`) that matches the position given in `ref_beg_seq_id`, which is a position within the protein sequence (not included in the data frame). `length` identifies the stretch of sequence for which positions match accordingly between structure and protein sequence. `entity_beg_seq_id` is a residue ID based on the standardised convention for mmCIF files.
- auth_seq_id: Residue identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure. This character vector has the same length as the `pdb_sequence` and each position is the identifier for the matching amino acid position in `pdb_sequence`. The contained values are not necessarily numbers and the values do not have to be positive.

Examples

```
pdb <- fetch_pdb(pdb_ids = c("6HG1", "1E9I", "6D3Q", "4JHW"))

head(pdb)
```

`fetch_pdb_structure` *Fetch PDB structure atom data from RCSB*

Description

Fetches atom data for a PDB structure from RCSB. If you want to retrieve metadata about PDB structures, use the function `fetch_pdb()`. The information retrieved is based on the .cif file of the structure, which may vary from the .pdb file.

Usage

```
fetch_pdb_structure(pdb_ids, return_data_frame = FALSE, show_progress = TRUE)
```

Arguments

pdb_ids	a character vector of PDB identifiers.
return_data_frame	a logical value that indicates if a data frame instead of a list is returned. It is recommended to only use this if not many pdb structures are retrieved. Default is FALSE.
show_progress	a logical value that indicates if a progress bar will be shown. Default is TRUE.

Value

A list that contains atom data for each PDB structures provided. If return_data_frame is TRUE, a data frame with this information is returned instead. The data frame contains the following columns:

- label_id: Uniquely identifies every atom in the structure following the standardised convention for mmCIF files. Example value: "5", "C12", "Ca3g28", "Fe3+17", "H*251", "boron2a", "C a phe 83 a 0", "Zn Zn 301 A 0"
- type_symbol: The code used to identify the atom species representing this atom type. Normally this code is the element symbol. The code may be composed of any character except an underscore with the additional proviso that digits designate an oxidation state and must be followed by a + or - character. Example values: "C", "Cu2+", "H(SDS)", "dummy", "FeNi".
- label_atom_id: Uniquely identifies every atom for the given residue following the standardised convention for mmCIF files. Example values: "CA", "HB1", "CB", "N"
- label_comp_id: A chemical identifier for the residue. For protein polymer entities, this is the three- letter code for the amino acid. For nucleic acid polymer entities, this is the one-letter code for the base. Example values: "ala", "val", "A", "C".
- label_asym_id: Chain identifier following the standardised convention for mmCIF files. Example values: "1", "A", "2B3".
- entity_id: Records details about the molecular entities that are present in the crystallographic structure. Usually all different types of molecular entities such as polymer entities, non-polymer entities or water molecules are numbered once for each structure. Each type of non-polymer entity has its own number. Thus, the highest number in this column represents the number of different molecule types in the structure.
- label_seq_id: Uniquely and sequentially identifies residues for each `label_asym_id`. This is always a number and the sequence of numbers always progresses in increasing numerical order.
- x: The x coordinate of the atom.
- y: The y coordinate of the atom.
- z: The z coordinate of the atom.
- site_occupancy: The fraction of the atom type present at this site.
- b_iso_or_equivalent: Contains the B-factor or isotopic atomic displacement factor for each atom.
- formal_charge: The net integer charge assigned to this atom. This is the formal charge assignment normally found in chemical diagrams. It is currently only assigned in a small subset of structures.

- auth_seq_id: An alternative residue identifier (`label_seq_id`) provided by the author of the structure in order to match the identification used in the publication that describes the structure. This does not need to be numeric and is therefore of type character.
- auth_comp_id: An alternative chemical identifier (`label_comp_id`) provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- auth_asym_id: An alternative chain identifier (`label_asym_id`) provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- pdb_model_number: The PDB model number.
- pdb_id: The protein database identifier for the structure.

Examples

```
pdb_structure <- fetch_pdb_structure(
  pdb_ids = c("6HG1", "1E9I", "6D3Q", "4JHW"),
  return_data_frame = TRUE
)

head(pdb_structure, n = 10)
```

`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	a character vector of metadata columns that should be imported from UniProt (all possible columns can be found here.)
batchsize	a numeric value that specifies the number of proteins processed in a single single query. Default is 200.
show_progress	a logical value that determines if a progress bar will be shown. Default is TRUE.

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 value that specifies the NCBI taxonomy identifier (TaxId) for and organism.
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 value that determines if 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`	a data frame that contains at least the input variables.
`grouping`	a character column in the `data` data frame that contains the grouping variable that can be either precursors, peptides or proteins.
`condition`	a character or numeric column in the `data` data frame that contains information on the sample condition.
`log2_intensity`	a numeric column in the `data` data frame that contains log2 transformed intensities.
`cv_limit`	optional, a numeric value that specifies the CV cutoff that will be applied. Default is 0.25.
`min_conditions`	a numeric value that specifies the minimum number of conditions for which grouping CVs should be below the cutoff.
`silent`	a logical value that specifies if a message with the number of filtered out conditions should be returned. Default is FALSE.

Value

The CV filtered data frame.

Examples

```
set.seed(123) # Makes example reproducible

# Create synthetic data
data <- create_synthetic_data(
  n_proteins = 50,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random",
  additional_metadata = FALSE
)

# Filter coefficients of variation
data_filtered <- filter_cv(
  data = data,
  grouping = peptide,
  condition = condition,
  log2_intensity = peptide_intensity_missing,
  cv_limit = 0.25,
  min_conditions = 2
)
```

---

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, ids, types = "is_a")
```

Arguments

data	a data frame that contains relational 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)`.
ids	a character vector of ChEBI IDs for which sub IDs should be searched.
types	a character vector containing the types of relationships that should be considered for the search. 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 double vectors 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

Search for chebi IDs that match a specific name pattern. A list of corresponding ChEBI IDs 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. It contains one element per 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 that contains at least the protein and peptide sequence.
`protein_sequence`	a character column in the `data` data frame that contains the protein sequence.
`peptide_sequence`	a character column in the `data` data frame that contains the peptide sequence.

Value

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

Examples

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

# Find peptide
find_peptide(
  data = data,
  protein_sequence = protein_sequence,
  peptide_sequence = peptide_sequence
)
```

`find_peptide_in_structure` *Finds peptide positions in a PDB structure based on positional matching*

Description

Finds peptide positions in a PDB structure. Often positions of peptides in UniProt and a PDB structure are different due to different lengths of structures. This function maps a peptide based on its UniProt positions onto a PDB structure. This method is superior to sequence alignment of the peptide to the PDB structure sequence, since it can also match the peptide if there are truncations or mismatches. This function also provides an easy way to check if a peptide is present in a PDB structure.

Usage

```
find_peptide_in_structure(
  peptide_data,
  peptide,
  start,
```

```
  end,
  uniprot_id,
  pdb_data = NULL,
  retain_columns = NULL
)
```

Arguments

peptide_data	a data frame containing at least the input columns to this function.
peptide	a character column in the peptide_data data frame that contains the sequence or any other unique identifier for the peptide that should be found.
start	a numeric column in the peptide_data data frame that contains start positions of peptides.
end	a numeric column in the peptide_data data frame that contains end positions of peptides.
uniprot_id	a character column in the peptide_data data frame that contains UniProt identifiers that correspond to the peptides.
pdb_data	optional, a data frame containing data obtained with fetch_pdb(). If not provided, information is fetched automatically. If this function should be run multiple times it is faster to fetch the information once and provide it to the function. If provided, make sure that the column names are identical to the ones that would be obtained by calling fetch_pdb().
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 peptide positions in the corresponding PDB structures. If a peptide is not found in any structure or no structure is associated with the protein, the data frame contains NAs values for the output columns. The data frame contains the following and additional columns:

- auth_asym_id: Chain identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure.
- label_asym_id: Chain identifier following the standardised convention for mmCIF files.
- peptide_seq_in_pdb: The sequence of the peptide mapped to the structure. If the peptide only maps partially, then only the part of the sequence that maps on the structure is returned.
- fit_type: The fit type is either "partial" or "fully" and it indicates if the complete peptide or only part of it was found in the structure.
- label_seq_id_start: Contains the first residue position of the peptide in the structure following the standardised convention for mmCIF files.
- label_seq_id_end: Contains the last residue position of the peptide in the structure following the standardised convention for mmCIF files.

- auth_seq_id_start: Contains the first residue position of the peptide in the structure based on the alternative residue identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure. This does not need to be numeric and is therefore of type character.
- auth_seq_id_end: Contains the last residue position of the peptide in the structure based on the alternative residue identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure. This does not need to be numeric and is therefore of type character.
- auth_seq_id: Contains all positions (separated by ";") of the peptide in the structure based on the alternative residue identifier provided by the author of the structure in order to match the identification used in the publication that describes the structure. This does not need to be numeric and is therefore of type character.
- n_peptides: The number of peptides from one protein that were searched for within the current structure.
- n_peptides_in_structure: The number of peptides from one protein that were found within the current structure.

Examples

```
# Create example data
peptide_data <- data.frame(
  uniprot_id = c("P0A8T7", "P0A8T7", "P60906"),
  peptide_sequence = c(
    "SGIVSFGKETKGKRRLVITPVDGSDPYEEMIPKWRQLNV",
    "NVFEGERVER",
    "AIGEVTDVVEKE"
  ),
  start = c(1160, 1197, 55),
  end = c(1198, 1206, 66)
)

# Find peptides in protein structure
peptide_in_structure <- find_peptide_in_structure(
  peptide_data = peptide_data,
  peptide = peptide_sequence,
  start = start,
  end = end,
  uniprot_id = uniprot_id
)

head(peptide_in_structure, n = 10)
```

`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 that contains at least the input variables.
sample	a character column in the data data frame that contains the sample names.
grouping	a character column in the data data frame that contains the precursor, peptide or protein identifiers.
response	a numeric column in the data data frame that contains the response values, e.g. log2 transformed intensities.
dose	a numeric column in the data data frame that contains the dose values, e.g. the treatment concentrations.
filter	a character value that determines 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	a numeric value which similar to completenss_MAR of the assign_missingness function 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`
: a numeric value which 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 that specifies the correlation cutoff used for data filtering.

`log_logarithmic`
: a logical value that indicates 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 value that indicates if model fit objects should be exported. These are usually very large and not necessary for further analysis.

`retain_columns`
: a vector that specifies columns that 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% of total replicates (adjusted downward) for at least 50% of 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 t he 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

```
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 2,
  frac_change = 1,
  n_replicates = 3,
  n_conditions = 8,
  method = "dose_response",
  concentrations = c(0, 1, 10, 50, 100, 500, 1000, 5000),
  additional_metadata = FALSE
)

# Perform dose response curve fit
drc_fit <- fit_drc_4p(
  data = data,
  sample = sample,
  grouping = peptide,
  response = peptide_intensity_missing,
  dose = concentration,
```

```
  retain_columns = c(protein, change_peptide)
)

glimpse(drc_fit)

head(drc_fit, n = 10)
```

---

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_log2,
  condition,
  comparison = comparison,
  missingness = missingness,
  noise = NULL,
  method = "ludovic",
  skip_log2_transform_error = FALSE,
  retain_columns = NULL
)
```

Arguments

data	a data frame 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	a character column in the data data frame that contains the sample names.
grouping	a character column in the data data frame that contains the precursor or peptide identifiers.
intensity_log2	a numeric column in the data data frame that contains the intensity values.
condition	a character or numeric column in the data data frame that contains the the conditions.
comparison	a character column in the data data frame that contains the the comparisons of treatment/reference pairs. This is an output of the assign_missingnes function.

missingness — a character column in the data data frame that contains the missingness type of the data determines how values for imputation are sampled. This should at least contain "MAR" or "MNAR". Missingness assigned as NA will not be imputed.

noise — a numeric column in the data data frame that contains the noise value for the precursor/peptide. Is only required if method = "noise". Note: Noise values need to be log2 transformed.

method — a character value that specifies 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. Both methods impute MAR data using the mean and variance of the condition with the missing data.

skip_log2_transform_error — a logical value that determines if 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 returned.

retain_columns — a vector that indicates columns that 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 10,
  frac_change = 0.5,
  n_replicates = 4,
  n_conditions = 2,
  method = "effect_random",
  additional_metadata = FALSE
)

head(data, n = 24)

# Assign missingness information
data_missing <- assign_missingness(
  data,
```

```
  sample = sample,
  condition = condition,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  ref_condition = "all",
  retain_columns = c(protein, peptide_intensity)
)

head(data_missing, n = 24)

# Perform imputation
data_imputed <- impute(
  data_missing,
  sample = sample,
  grouping = peptide,
  intensity_log2 = peptide_intensity_missing,
  condition = condition,
  comparison = comparison,
  missingness = missingness,
  method = "ludovic",
  retain_columns = c(protein, peptide_intensity)
)

head(data_imputed, n = 24)
```

`map_peptides_on_structure` *Maps peptides onto a PDB structure or AlphaFold prediction*

Description

Peptides are mapped onto PDB structures or AlphaFold prediction based on their positions. This is accomplished by replacing the B-factor information in the structure file with values that allow highlighting of peptides, protein regions or amino acids when the structure is coloured by B-factor. In addition to simply highlighting peptides, protein regions or amino acids, a continuous variable such as fold changes associated with them can be mapped onto the structure as a colour gradient.

Usage

```
map_peptides_on_structure(
  peptide_data,
  uniprot_id,
  pdb_id,
  chain,
  auth_seq_id,
  map_value,
  file_format = ".cif",
  scale_per_structure = TRUE,
  export_location = NULL,
```

```
  structure_file = NULL,
  show_progress = TRUE
)
```

Arguments

peptide_data	a data frame that contains the input columns to this function. If structure or prediction files should be fetched automatically, please provide column names to the following arguments: **uniprot_id**, **pdb_id**, **chain**, **auth_seq_id**, **map_value**. If no PDB structure for a protein is available the pdb_id and chain column should contain NA at these positions. If a structure or prediction file is provided in the structure_file argument, this data frame should only contain information associated with the provided structure. In case of a user provided structure, column names should be provided to the following arguments: **uniprot_id**, **chain**, **auth_seq_id**, **map_value**.
uniprot_id	a character column in the peptide_data data frame that contains UniProt identifiers for a corresponding peptide, protein region or amino acid.
pdb_id	a character column in the peptide_data data frame that contains PDB identifiers for structures in which a corresponding peptide, protein region or amino acid is found. If a protein prediction should be fetched from AlphaFold, this column should contain NA. This column is not required if a structure or prediction file is provided in the structure_file argument.
chain	a character column in the peptide_data data frame that contains the name of the chain from the PDB structure in which the peptide, protein region or amino acid is found. If a protein prediction should be fetched from AlphaFold, this column should contain NA. If an AlphaFold prediction is provided to the structure_file argument the chain should be provided as usual (All AlphaFold predictions only have chain A). **Important:** please provide the author defined chain definitions for both ".cif" and ".pdb" files. When the output of the find_peptide_in_structure function is used as the input for this function, this corresponds to the auth_asym_id column.
auth_seq_id	optional, a character (or numeric) column in the peptide_data data frame that contains semicolon separated positions of peptides, protein regions or amino acids in the corresponding PDB structure or AlphaFold prediction. This information can be obtained from the find_peptide_in_structure function. The corresponding column in the output is called auth_seq_id. In case of AlphaFold predictions, UniProt positions should be used. If signal positions and not stretches of amino acids are provided, the column can be numeric and does not need to contain the semicolon separator.
map_value	a numeric column in the peptide_data data frame that contains a value associated with each peptide, protein region or amino acid. If one start to end position pair has multiple different map values, the maximum will be used. This value will be displayed as a colour gradient when mapped onto the structure. The value can for example be the fold change, p-value or score associated with each peptide, protein region or amino acid (selection). If the selections should be displayed with just one colour, the value in this column should be the same for every selection. For the mapping, values are scaled between 50 and 100.

Regions in the structure that do not map any selection receive a value of 0. If an amino acid position is associated with multiple mapped values, e.g. from different peptides, the maximum mapped value will be displayed.

`file_format` a character vector containing the file format of the structure that will be fetched from the database for the PDB identifiers provided in the `pdb_id` column. This can be either ".cif" or ".pdb". The default is `".cif"`. We recommend using ".cif" files since every structure contains a ".cif" file but not every structure contains a ".pdb" file. Fetching and mapping onto ".cif" files takes longer than for ".pdb" files. If a structure file is provided in the `structure_file` argument, the file format is detected automatically and does not need to be provided.

`scale_per_structure`
a logical value that specifies if scaling should be performed for each structure independently (TRUE) or over the whole data set (FALSE). The default is TRUE, which scales the scores of each structure independently so that each structure has a score range from 50 to 100.

`export_location`
optional, a character argument specifying the path to the location in which the fetched and altered structure files should be saved. If left empty, they will be saved in the current working directory. The location should be provided in the following format "folderA/folderB".

`structure_file` optional, a character argument specifying the path to the location and name of a structure file in ".cif" or ".pdb" format. If a structure is provided the `peptide_data` data frame should only contain mapping information for this structure.

`show_progress` a logical, if `show_progress = TRUE`, a progress bar will be shown (default is TRUE).

Value

The function exports a modified ".pdb" or ".cif" structure file. B-factors have been replaced with scaled (50-100) values provided in the `map_value` column.

Examples

```
# Load libraries
library(dplyr)

# Create example data
peptide_data <- data.frame(
  uniprot_id = c("P0A8T7", "P0A8T7", "P60906"),
  peptide_sequence = c(
    "SGIVSFGKETKGKRRLVITPVDGSDPYEEMIPKWRQLNV",
    "NVFEGERVER",
    "AIGEVTDVVEKE"
  ),
  start = c(1160, 1197, 55),
  end = c(1198, 1206, 66),
```

```
  map_value = c(70, 100, 100)
)

# Find peptide positions in structures
positions_structure <- find_peptide_in_structure(
  peptide_data = peptide_data,
  peptide = peptide_sequence,
  start = start,
  end = end,
  uniprot_id = uniprot_id,
  retain_columns = c(map_value)) %>%
  filter(pdb_ids %in% c("6UU2", "2EL9"))

# Map peptides on structures
# You can determine the preferred output location
# with the export_location argument. Currently it
# is saved in the working directory.
map_peptides_on_structure(
  peptide_data = positions_structure,
  uniprot_id = uniprot_id,
  pdb_id = pdb_ids,
  chain = auth_asym_id,
  auth_seq_id = auth_seq_id,
  map_value = map_value,
  file_format = ".pdb",
  export_location = getwd()
)
```

`normalise` *Intensity normalisation*

Description

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

Usage

```
normalise(data, sample, intensity_log2, method = "median")
```

Arguments

`data`	a data frame containing at least sample names and intensity values.
`sample`	a character column in the `data` data frame that contains the sample names.

intensity_log2	a numeric column in the data data frame that contains the log2 transformed intensity values to be normalised.
method	a character value specifying the method to be used for normalisation. Default is "median".

Value

A data frame 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)
)

normalise(data,
  sample = r_file_name,
  intensity_log2 = intensity_log2,
  method = "median"
)
```

---

parallel_create_structure_contact_map

Creates a contact map of all atoms from a structure file (using parallel processing)

Description

This function is a wrapper around create_structure_contact_map() that allows the use of all system cores for the creation of contact maps. Alternatively, it can be used for sequential processing of large datasets. The benefit of this function over create_structure_contact_map() is that it processes contact maps in batches, which is recommended for large datasets. If used for parallel processing 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(multisession) or automatically by the function (maximum number of workers is 12 in this case). If workers are set up manually the processing_type argument should be set to "parallel manual". In this case workers can be terminated after completion with future::plan(sequential).

Usage

```
parallel_create_structure_contact_map(
  data,
  data2 = NULL,
  id,
  chain = NULL,
  auth_seq_id = NULL,
```

```
  distance_cutoff = 10,
  pdb_model_number_selection = c(0, 1),
  return_min_residue_distance = TRUE,
  export = FALSE,
  export_location = NULL,
  split_n = 40,
  processing_type = "parallel"
)
```

Arguments

data	a data frame containing at least a column with PDB ID information of which the name can be provided to the id argument. If only this column is provided, all atom or residue distances are calculated. Additionally, a chain column can be present in the data frame of which the name can be provided to the chain argument. If chains are provided, only distances of this chain relative to the rest of the structure are calculated. Multiple chains can be provided in multiple rows. If chains are provided for one structure but not for another, the rows should contain NAs. Furthermore, specific residue positions can be provided in the auth_seq_id column if the selection should be further reduced. It is not recommended to create full contact maps for more than a few structures due to time and memory limitations. If contact maps are created only for small regions it is possible to create multiple maps at once. By default distances of regions provided in this data frame to the complete structure are computed. If distances of regions from this data frame to another specific subset of regions should be computed, the second subset of regions can be provided through the optional data2 argument.
data2	optional, a data frame that contains a subset of regions for which distances to regions provided in the data data frame should be computed. If regions from the data data frame should be compared to the whole structure, data2 does not need to be provided. This data frame should have the same structure and column names as the data data frame.
id	a character column in the data data frame that contains PDB or UniProt IDs for structures or AlphaFold predictions of which contact maps should be created. If a structure not downloaded directly from PDB is provided (i.e. a locally stored structure file) to the structure_file argument, this column should contain "my_structure" as content.
chain	optional, a character column in the data data frame that contains chain identifiers for the structure file. Identifiers defined by the structure author should be used. Distances will be only calculated between the provided chains and the rest of the structure.
auth_seq_id	optional, a character (or numeric) column in the data data frame that contains semicolon separated positions of regions for which distances should be calculated. This always needs to be provided in combination with a corresponding chain in chain. The position should match the positioning defined by the structure author. For PDB structures this information can be obtained from the find_peptide_in_structure function. The corresponding column in the output is called auth_seq_id. If an AlphaFold prediction is provided, UniProt

positions should be used. If single positions and not stretches of amino acids are provided, the column can be numeric and does not need to contain the semicolon separator.

`distance_cutoff`
: a numeric value specifying the distance cutoff in Angstrom. All values for pairwise comparisons are calculated but only values smaller than this cutoff will be returned in the output. If a cutoff of e.g. 5 is selected then only residues with a distance of 5 Angstrom and less are returned. Using a small value can reduce the size of the contact map drastically and is therefore recommended. The default value is 10.

`pdb_model_number_selection`
: a numeric vector specifying which models from the structure files should be considered for contact maps. E.g. NMR models often have many models in one file. The default for this argument is c(0, 1). This means the first model of each structure file is selected for contact map calculations. For AlphaFold predictions the model number is 0 (only .pdb files), therefore this case is also included here.

`return_min_residue_distance`
: a logical value that specifies if the contact map should be returned for all atom distances or the minimum residue distances. Minimum residue distances are smaller in size. If atom distances are not strictly needed it is recommended to set this argument to TRUE. The default is TRUE.

`export`
: a logical value that indicates if contact maps should be exported as ".csv". The name of the file will be the structure ID. Default is `export = FALSE`.

`export_location`
: optional, a character value that specifies the path to the location in which the contact map should be saved if `export = TRUE`. If left empty, they will be saved in the current working directory. The location should be provided in the following format "folderA/folderB".

`split_n`
: a numeric value that specifies the number of structures that should be included in each batch. Default is 40.

`processing_type`
: a character value that is either "parallel" for parallel processing or "sequential" for sequential processing. Alternatively it can also be "parallel manual" in this case you have to set up the number of cores on your own using the `future::plan(multisession)` function. The default is "parallel".

Value

A list of contact maps for each PDB or UniProt ID provided in the input is returned. If the `export` argument is TRUE, each contact map will be saved as a ".csv" file in the current working directory or the location provided to the `export_location` argument.

Examples

```
## Not run:
# Create example data
data <- data.frame(
  pdb_id = c("6NPF", "1C14", "3NIR"),
```

```
  chain = c("A", "A", NA),
  auth_seq_id = c("1;2;3;4;5;6;7", NA, NA)
)

# Create contact map
contact_maps <- parallel_create_structure_contact_map(
  data = data,
  id = pdb_id,
  chain = chain,
  auth_seq_id = auth_seq_id,
  split_n = 1,
)

str(contact_maps[["3NIR"]])

contact_maps

## 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(multisession)` 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 that contains at least the input variables.
sample	a character column in the data data frame that contains the sample names.
grouping	a character column in the data data frame that contains the precursor, peptide or protein identifiers.
response	a numeric column in the data data frame that contains the response values, e.g. log2 transformed intensities.
dose	a numeric column in the data data frame that contains the dose values, e.g. the treatment concentrations.
filter	a character value that determines if models should be filtered and if they 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	a numeric value which similar to completenss_MAR of the assign_missingness function 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	a numeric value which 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 that specifies the correlation cutoff used for data filtering.
log_logarithmic	a logical value that indicates 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 that specifies columns that 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, a numeric value that specifies 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% of total replicates (adjusted downward) for at least 50% of 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:
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible
```

```
# Create example data
data <- create_synthetic_data(
  n_proteins = 2,
  frac_change = 1,
  n_replicates = 3,
  n_conditions = 8,
  method = "dose_response",
  concentrations = c(0, 1, 10, 50, 100, 500, 1000, 5000),
  additional_metadata = FALSE
)

# Perform dose response curve fit
drc_fit <- parallel_fit_drc_4p(
  data = data,
  sample = sample,
  grouping = peptide,
  response = peptide_intensity_missing,
  dose = concentration,
  retain_columns = c(protein, change_peptide)
)

glimpse(drc_fit)

head(drc_fit, n = 10)

## End(Not run)
```

`peptide_profile_plot` *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

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

Arguments

data	a data frame that contains at least the input variables.
sample	a character column in the data data frame that contains sample names.
peptide	a character column in the data data frame that contains peptide or precursor names.
intensity_log2	a numeric column in the data data frame that contains log2 transformed intensities.
grouping	a character column in the data data frame that contains groups by which the data should be split. This can be for example protein IDs.
targets	a character vector that specifies 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	a logical value. 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 value that indicates 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 value that indicates 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 that provides the name of the exported file if export = TRUE.

Value

A list of peptide profile plots.

Examples

```
# Create example data
data <- data.frame(
  sample = c(
    rep("S1", 6),
    rep("S2", 6),
    rep("S1", 2),
    rep("S2", 2)
  ),
  protein_id = c(
    rep("P1", 12),
    rep("P2", 4)
  ),
  precursor = c(
    rep(c("A1", "A2", "B1", "B2", "C1", "D1"), 2),
    rep(c("E1", "F1"), 2)
```

```
  ),
  peptide = c(
    rep(c("A", "A", "B", "B", "C", "D"), 2),
    rep(c("E", "F"), 2)
  ),
  intensity = c(
    rnorm(n = 6, mean = 15, sd = 2),
    rnorm(n = 6, mean = 21, sd = 1),
    rnorm(n = 2, mean = 15, sd = 1),
    rnorm(n = 2, mean = 15, sd = 2)
  )
)

# Calculate protein abundances and retain precursor
# abundances that can be used in a peptide profile plot
complete_abundances <- calculate_protein_abundance(
  data,
  sample = sample,
  protein_id = protein_id,
  precursor = precursor,
  peptide = peptide,
  intensity_log2 = intensity,
  method = "sum",
  for_plot = TRUE
)

# Plot protein abundance profile
# protein_abundance_plot can be set to
# FALSE to to also colour precursors
peptide_profile_plot(
  data = complete_abundances,
  sample = sample,
  peptide = precursor,
  intensity_log2 = intensity,
  grouping = protein_id,
  targets = c("P1"),
  protein_abundance_plot = TRUE
)
```

`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.

`ptsi_pgk` *Structural analysis example data*

Description

Example data used for the vignette about structural analysis. The data was obtained from Cappelletti 2021 and corresponds to two separate experiments. Both experiments were limited proteolyis coupled to mass spectrometry (LiP-MS) experiments conducted on purified proteins. The first protein is phosphoglycerate kinase 1 (pgk) and it was treated with 25mM 3-phosphoglyceric acid (3PG). The second protein is phosphoenolpyruvate-protein phosphotransferase (ptsI) and it was treated with 25mM fructose 1,6-bisphosphatase (FBP). From both experiments only peptides belonging to either protein were used for this data set. The ptsI data set contains precursor level data while the pgk data set contains peptide level data. The pgk data can be obtained from supplementary table 3 from the tab named "pgk+3PG". The ptsI data is only included as raw data and was analysed using the functions of this package.

Usage

```
ptsi_pgk
```

Format

A data frame containing differential abundances and adjusted p-values for peptides/precursors of two proteins.

Source

Cappelletti V, Hauser T, Piazza I, Pepelnjak M, Malinovska L, Fuhrer T, Li Y, Dörig C, Boersema P, Gillet L, Grossbach J, Dugourd A, Saez-Rodriguez J, Beyer A, Zamboni N, Caflisch A, de Souza N, Picotti P. Dynamic 3D proteomes reveal protein functional alterations at high resolution in situ. Cell. 2021 Jan 21;184(2):545-559.e22. doi: 10.1016/j.cell.2020.12.021. Epub 2020 Dec 23. PMID: 33357446; PMCID: PMC7836100.

pval_distribution_plot *Plot histogram of p-value distribution*

Description

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

Usage

```
pval_distribution_plot(data, grouping, pval, facet_by = NULL)
```

Arguments

data	a data frame that contains at least grouping identifiers (precursor, peptide or protein) and p-values derived from any statistical test.
grouping	a character column in the data data frame that contains either precursor, peptide or protein identifiers. For each entry in this column there should be one unique p-value. That means the statistical test that created the p-value should have been performed on the level of the content of this column.
pval	a numeric column in the data data frame that contains p-values.
facet_by	optional, a character column that contains information by which the data should be faceted into multiple plots.

Value

A histogram plot that shows the p-value distribution.

Examples

```
set.seed(123) # Makes example reproducible

# Create example data
data <- data.frame(
  peptide = paste0("peptide", 1:1000),
  pval = runif(n = 1000)
)

# Plot p-values
pval_distribution_plot(
  data = data,
  grouping = peptide,
  pval = pval
)
```

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 that contains at least sample names, peptide or precursor identifiers and missed cleavage counts for each peptide or precursor.
sample	a character column in the data data frame that contains the sample name.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
charge_states	a character or numeric column in the data data frame that contains the different charge states assigned to the precursor or peptide.
intensity	a numeric column in the data data frame that contains 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	a logical value that specifies 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	a character value that indicates 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	a logical value that indicates whether the result should be plotted.
interactive	a logical value that specifies 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

```
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
) %>%
  mutate(intensity_non_log2 = 2^peptide_intensity_missing)

# Calculate charge percentages
qc_charge_states(
  data = data,
  sample = sample,
  grouping = peptide,
  charge_states = charge,
  intensity = intensity_non_log2,
  method = "intensity",
  plot = FALSE
)

# Plot charge states
qc_charge_states(
  data = data,
  sample = sample,
  grouping = peptide,
  charge_states = charge,
  intensity = intensity_non_log2,
  method = "intensity",
  plot = TRUE
)
```

---

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 that contains at least the input variables.
sample	a character column in the data data frame that contains the sample names.
protein	a character column in the data data frame that contains protein IDs or protein names.
is_contaminant	a logical column that indicates if the protein is a contaminant.
intensity	a numeric column in the data data frame that contains the corresponding raw or normalised intensity values (not log2).
n_contaminants	a numeric value that indicates how many contaminants should be displayed individually. The rest is combined to a group called "other". The default is 5.
plot	a logical value that indicates if a plot is returned. If FALSE a table is returned.
interactive	a logical value that indicates if the plot is made interactive using the r package plotly.

Value

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

Examples

```
data <- data.frame(
  sample = c(rep("sample_1", 10), rep("sample_2", 10)),
 leading_razor_protein = c(rep(c("P1", "P1", "P1", "P2", "P2", "P2", "P2", "P3", "P3", "P3"), 2)),
  potential_contaminant = c(rep(c(rep(TRUE, 7), rep(FALSE, 3)), 2)),
  intensity = c(rep(1, 2), rep(4, 4), rep(6, 4), rep(2, 3), rep(3, 5), rep(4, 2))
)

qc_contaminants(
  data,
  sample = sample,
  protein = leading_razor_protein,
  is_contaminant = potential_contaminant,
  intensity = intensity
)
```

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	a character column in the data data frame that contains the grouping variables (e.g. peptides, precursors or proteins).
condition	a column in the data data frame that contains condition information (e.g. "treated" and "control").
intensity	a numeric column in the data data frame that contains the corresponding raw or untransformed normalised intensity values for each peptide or precursor.
plot	a logical value that indicates whether the result should be plotted.
plot_style	a character value that indicates 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 a data frame with the median CVs in % or a plot showing the distribution of the CVs is returned.

Examples

```
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible
```

```
# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
) %>%
  mutate(intensity_non_log2 = 2^peptide_intensity_missing)

# Calculate coefficients of variation
qc_cvs(
  data = data,
  grouping = peptide,
  condition = condition,
  intensity = intensity_non_log2,
  plot = FALSE
)

# Plot coefficients of variation
# Different plot styles are available
qc_cvs(
  data = data,
  grouping = peptide,
  condition = condition,
  intensity = intensity_non_log2,
  plot = TRUE,
  plot_style = "violin"
)
```

---

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	a character column in the data data frame that contains the sample names.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
intensity	a numeric column in the data data frame that contains any intensity intensity values that missingness should be determined for.
digestion	optional, a character column in the data data frame that indicates the mode of digestion (limited proteolysis or tryptic digest). Alternatively, any other variable by which the data should be split can be provided.
plot	a logical value that indicates whether the result should be plotted.
interactive	a logical value that specifies whether the plot should be interactive (default is FALSE).

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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Determine data completeness
qc_data_completeness(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  plot = FALSE
)

# Plot data completeness
qc_data_completeness(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  plot = TRUE
)
```

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	a character column in the data data frame that contains the sample name.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
intensity	a character column in the data data frame that contains raw or log2 transformed intensities. If remove_na_intensities = FALSE, this argument is optional.
remove_na_intensities	a logical value that specifies 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, a column in the data data frame that contains condition information (e.g. "treated" and "control"). If this column is provided, the bars in the plot will be coloured according to the condition.
title	optional, a character value that specifies the plot title (default is "ID count per sample").
plot	a logical value that indicates whether the result should be plotted.
interactive	a logical value that specifies 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Calculate number of identifications
qc_ids(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  condition = condition,
  plot = FALSE
)

# Plot number of identifications
qc_ids(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  condition = condition,
  plot = TRUE
)
```

---

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 that contains at least sample names, grouping identifiers (precursor, peptide or protein) and log2 transformed intensities for each grouping identifier.
sample	an optional character or factor column in the data data frame that contains the sample name. If the sample column is of type factor, the ordering is based on the factor levels. NOTE: If the overall distribution should be returned please do not provide the name of the sample column.
grouping	a character column in the data data frame that contains the grouping variables (e.g. peptides, precursors or proteins).
intensity_log2	a numeric column in the data data frame that contains the log2 transformed intensities of each grouping identifier sample combination.
plot_style	a character value that indicates 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Plot intensity distribution
# The plot style can be changed
qc_intensity_distribution(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity_log2 = peptide_intensity_missing,
  plot_style = "boxplot"
)
```

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 that contains at least the input variables.
sample	a character column in the data data frame that contains the sample name.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
intensity	a numeric column in the data data frame that contains intensity values. The intensity should be ideally log2 transformed, but also non-transformed values can be used.
plot	a logical value that indicates whether the result should be plotted.
interactive	a logical value that specifies whether the plot should be interactive (default is FALSE).

Value

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

Examples

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
```

```
  method = "effect_random"
)

# Calculate median intensities
qc_median_intensities(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  plot = FALSE
)

# Plot median intensities
qc_median_intensities(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  plot = TRUE
)
```

`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`	a character column in the `data` data frame that contains the sample name.

`grouping` a character column in the `data` data frame that contains either precursor or peptide identifiers.

`missed_cleavages` a numeric column in the `data` data frame that contains the counts of missed cleavages per peptide or precursor.

`intensity` a numeric column in the `data` data frame that contains 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` a logical value that specifies 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` a character value that indicates 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 value that indicates whether the result should be plotted.

`interactive` a logical value that specifies 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Calculate missed cleavage percentages
qc_missed_cleavages(
  data = data,
  sample = sample,
  grouping = peptide,
  missed_cleavages = n_missed_cleavage,
  intensity = peptide_intensity_missing,
  method = "intensity",
  plot = FALSE
)
```

```
# Plot missed cleavages
qc_missed_cleavages(
  data = data,
  sample = sample,
  grouping = peptide,
  missed_cleavages = n_missed_cleavage,
  intensity = peptide_intensity_missing,
  method = "intensity",
  plot = TRUE
)
```

---

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 that contains sample names, peptide or precursor identifiers, corresponding intensities and a condition column indicating e.g. the treatment.
sample	a character column in the data data frame that contains the sample name.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
intensity	a numeric column in the data data frame that contains the corresponding intensity values for each peptide or precursor.
condition	a column in the data data frame that contains condition information (e.g. "treated" and "control").
components	a 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, a character column in the data data frame that indicates the mode of digestion (limited proteolysis or tryptic digest). Alternatively, any other variable by which the data should be split can be provided.

plot_style — a character value that specifies 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 principal component analysis plot showing PC1 and PC2. If plot_style = "scree", a scree plot for all dimensions is returned.

Examples

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
)

# Plot scree plot
qc_pca(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  condition = condition,
  plot_style = "scree"
)

# Plot principal components
qc_pca(
  data = data,
  sample = sample,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  condition = condition
)
```

---

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`	a character column in the `data` data frame that contains the sample names.
`intensity`	a numeric column in the `data` data frame that contains intensities. If `remove_na_intensities = FALSE`, this argument is not required.
`retention_time`	a numeric column in the `data` data frame that contains retention times of precursors.
`peak_width`	a numeric column in the `data` data frame that contains peak width information. It is not required if `retention_time_start` and `retention_time_end` columns are provided.
`retention_time_start`	a numeric column in the `data` data frame that contains the start time of the precursor elution peak. It is not required if the `peak_width` column is provided.
`retention_time_end`	a numeric column in the `data` data frame that contains the end time of the precursor elution peak. It is not required if the `peak_width` column is provided.
`remove_na_intensities`	a logical value that specifies 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 value that specifies 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

```
data <- data.frame(
  r_file_name = c(rep("sample_1", 10), rep("sample2", 10)),
  fg_quantity = c(rep(2000, 20)),
  eg_mean_apex_rt = c(rep(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), 2)),
  eg_start_rt = c(0.5, 1, 3, 4, 5, 6, 7, 7.5, 8, 9, 1, 2, 2, 3, 4, 5, 5, 8, 9, 9),
  eg_end_rt = c(
    1.5, 2, 3.1, 4.5, 5.8, 6.6, 8, 8, 8.4,
    9.1, 3, 2.2, 4, 3.4, 4.5, 5.5, 5.6, 8.3, 10, 12
  )
)
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
)
```

---

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 that contains at least the input columns.
sample	a character column in the data data frame that contains the sample names.
peptide	a character column in the data data frame that contains the peptide sequence.

pep_type — a character column in the data data frame that contains the peptide type. Can be obtained using the find_peptide and assign_peptide_type function together.

intensity — a numeric column in the data data frame that contains 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 — a logical value that specifies 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 — a character value that indicates 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 value that indicates whether the result should be plotted.

interactive — a logical value that indicates 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

```
# Load libraries
library(dplyr)

set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
) %>%
  mutate(intensity_non_log2 = 2^peptide_intensity_missing)

# Determine peptide type percentages
qc_peptide_type(
  data = data,
  sample = sample,
  peptide = peptide,
  pep_type = pep_type,
  intensity = intensity_non_log2,
  method = "intensity",
  plot = FALSE
```

```
)

# Plot peptide type
qc_peptide_type(
  data = data,
  sample = sample,
  peptide = peptide,
  pep_type = pep_type,
  intensity = intensity_non_log2,
  method = "intensity",
  plot = TRUE
)
```

`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 t he fraction of detected proteins to all proteins in the proteome is calculated.

Usage

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

Arguments

`data`	a data frame that contains at least sample names and protein ID's.
`sample`	a character column in the `data` data frame that contains the sample name.
`protein_id`	a character or numeric column in the `data` data frame that contains protein identifiers such as UniProt accessions.
`organism_id`	a numeric value that specifies a NCBI taxonomy identifier (TaxId) of the organism used. Human: 9606, S. cerevisiae: 559292, E. coli: 83333.
`reviewed`	a logical value that determines if only reviewed protein entries will be considered as the full proteome. Default is TRUE.
`plot`	a logical value that specifies whether the result should be plotted.
`interactive`	a logical value that indicates 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

```
# Create example data
proteome <- data.frame(id = 1:4518)
data <- data.frame(
  sample = c(rep("A", 101), rep("B", 1000), rep("C", 1000)),
  protein_id = c(proteome$id[1:100], proteome$id[1:1000], proteome$id[1000:2000])
)

# Calculate proteome coverage
qc_proteome_coverage(
  data = data,
  sample = sample,
  protein_id = protein_id,
  organism_id = 83333,
  plot = FALSE
)

# Plot proteome coverage
qc_proteome_coverage(
  data = data,
  sample = sample,
  protein_id = protein_id,
  organism_id = 83333,
  plot = TRUE
)
```

---

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 data frame that contains at least the input variables.
sample	a character column in the data data frame that contains the sample names.
grouping	a character column in the data data frame that contains precursor or peptide identifiers.
intensity_log2	a numeric column in the data data frame that contains log2 intensity values.
condition	a character or numeric column in the data data frame that contains the conditions.
digestion	optional, a character column in the data data frame that contains information about the digestion method used. e.g. "LiP" or "tryptic control".
run_order	optional, a character or numeric column in the data data frame that contains the order in which samples were measured. Useful to investigate batch effects due to run order.
method	a character value that specifies the method to be used for correlation. "spearman" is the default but can be changed to "pearson" or "kendall".
interactive	a logical value that specifies whether the plot should be interactive. 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

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Create sample correlation heatmap
qc_sample_correlation(
  data = data,
  sample = sample,
```

```
  grouping = peptide,
  intensity_log2 = peptide_intensity_missing,
  condition = condition
)
```

---

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 that contains at least the input variables.
protein_identifier	a character column in the data data frame that contains protein identifiers.
coverage	a numeric column in the data data frame that contains protein coverage in percent. This information can be obtained using the sequence_coverage function.
sample	optional, a character column in the data data frame that contains 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 value that specifies whether the plot should be interactive (default is FALSE).

Value

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

See Also

sequence_coverage

Examples

```
set.seed(123) # Makes example reproducible

# Create example data
data <- create_synthetic_data(
  n_proteins = 100,
  frac_change = 0.05,
  n_replicates = 3,
  n_conditions = 2,
  method = "effect_random"
)

# Plot sequence coverage
qc_sequence_coverage(
  data = data,
  protein_identifier = protein,
  coverage = coverage
)
```

`randomise_queue` *Randomise samples in MS queue*

Description

[Experimental] 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 that contains a queue. If not provided a queue file can be chosen interactively.
`rows`	optional, a numeric vector that specifies a range of rows in for which samples should be randomized.
`export`	a logical value that determines if 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

```
queue <- 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\\DIA_120min",
  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"
)

head(queue, n = 20)

randomised_queue <- randomise_queue(
  data = queue,
  export = FALSE
)

head(randomised_queue, n = 20)
```

---

rapamycin_10uM *Rapamycin 10 uM example data*

Description

Rapamycin example data used for the vignette about binary control/treated data. The data was 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 was 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`	a character value that specifies 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 value that contains the protein sequence.
positions_start	a numeric vector of start positions of the identified peptides.
positions_end	a numeric 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. If a vector containing only one value or repeatedly the same value is provided, 1 is returned as the scaled value for method = "01" and 0 is returned for metod = "center".

Usage

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

Arguments

x	a numeric vector
method	a character value that specifies 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")
```

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.

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,
  type = "text/tab-separated-values",
  ...
)
```

Arguments

url	a character value of an URL to the website that contains the table that should be downloaded.
max_tries	a numeric value that specifies the number of times the function tries to download the data in case an error occurs.
silent	a logical value that specifies if individual messages are printed after each try that failed.
type	a character value that specifies the type of data at the target URL. Options are all options that can be supplied to httr::content, these include e.g. "text/tab-separated-values", "application/json" and "txt/csv". Default is "text/tab-separated-values". Default is "tab-separated-values".
...	other parameters supplied to the parsing function used by httr::content.

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	a numeric vector that contains the means of group1.
mean2	a numeric vector that contains the means of group2.
sd1	a numeric vector that contains the standard deviations of group1.
sd2	a numeric vector that contains the standard deviations of group2.
n1	a numeric vector that contains the number of replicates used for the calculation of each mean and standard deviation of group1.
n2	a numeric vector that contains the number of replicates used for the calculation of each mean and standard deviation of group2.
log_values	a logical value that indicates 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
)
```

---

viridis_colours *Viridis colour scheme*

Description

A colour scheme by the viridis colour scheme from the viridis R package.

Usage

```
viridis_colours
```

Format

A vector containing 256 colours

Source

viridis R package

---

volcano_plot *Volcano plot*

Description

Plots a volcano plot for the given input.

Usage

```
volcano_plot(
  data,
  grouping,
  log2FC,
  significance,
  method,
  target_column = NULL,
  target = NULL,
```

```
  facet_by = NULL,
  facet_scales = "fixed",
  title = "Volcano plot",
  x_axis_label = "log2(fold change)",
  y_axis_label = "-log10(p-value)",
  legend_label = "Target",
  log2FC_cutoff = 1,
  significance_cutoff = 0.01,
  interactive = FALSE
)
```

Arguments

data	a data frame that contains at least the input variables.
grouping	a character column in the data data frame that contains either precursor or peptide identifiers.
log2FC	a character column in the data data frame that contains the log2 transfromed fold changes between two conditions.
significance	a character column in the data data frame that contains the p-value or adjusted p-value for the corresponding fold changes. The values in this column will be transformed using the -log10 and displayed on the y-axis of the plot.
method	a character value that specifies 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, a 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, a vector 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, a character column that contains information by which the data should be faceted into multiple plots.
facet_scales	a character value that specifies if the scales should be "free", "fixed", "free_x" or "free_y", if a faceted plot is created. These inputs are directly supplied to the scales argument of ggplot2::facet_wrap().
title	optional, a character value that specifies the title of the volcano plot. Default is "Volcano plot".
x_axis_label	optional, a character value that specifies the x-axis label. Default is "log2(fold change)".
y_axis_label	optional, a character value that specifies the y-axis label. Default is "-log10(q-value)".
legend_label	optional, a character value that specifies the legend label. Default is "Target".

log2FC_cutoff
: optional, a numeric value that specifies the log2 transformed fold change cutoff used for the vertical lines, which can be used to assess the significance of changes. Default value is 1.

significance_cutoff
: optional, a character vector that specifies the p-value cutoff used for the horizontal cutoff line, which can be used to assess the significance of changes. The vector can consist solely of one element, which is the cutoff value. In that case the cutoff will be applied directly to the plot. Alternatively, a second element can be provided to the vector that specifies a column in the `data` data frame which contains e.g. adjusted p-values. In that case the y-axis of the plot could display p-values that are provided to the `significance` argument, while the horizontal cutoff line is on the scale of adjusted p-values transformed to the scale of p-values. The provided vector can be e.g. `c(0.05, "adj_pval")`. In that case the function looks for the closest adjusted p-value above and below 0.05 and takes the mean of the corresponding p-values as the cutoff line. If there is no adjusted p-value in the data that is below 0.05 no line is displayed. This allows the user to display volcano plots using p-values while using adjusted p-values for the cutoff criteria. This is often preferred because adjusted p-values are related to unadjusted p-values often in a complex way that makes them hard to be interpret when plotted. Default is `c(0.01)`.

interactive
: a logical value that specifies whether the plot should be interactive (default is 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

```
set.seed(123) # Makes example reproducible

# Create synthetic data
data <- create_synthetic_data(
  n_proteins = 10,
  frac_change = 0.5,
  n_replicates = 4,
  n_conditions = 3,
  method = "effect_random",
  additional_metadata = FALSE
)

# Assign missingness information
data_missing <- assign_missingness(
  data,
  sample = sample,
  condition = condition,
  grouping = peptide,
  intensity = peptide_intensity_missing,
  ref_condition = "all",
```

```
  retain_columns = c(protein, change_peptide)
)

# Calculate differential abundances
diff <- calculate_diff_abundance(
  data = data_missing,
  sample = sample,
  condition = condition,
  grouping = peptide,
  intensity_log2 = peptide_intensity_missing,
  missingness = missingness,
  comparison = comparison,
  method = "t-test",
  retain_columns = c(protein, change_peptide)
)

volcano_plot(
  data = diff,
  grouping = peptide,
  log2FC = diff,
  significance = pval,
  method = "target",
  target_column = change_peptide,
  target = TRUE,
  facet_by = comparison,
  significance_cutoff = c(0.05, "adj_pval")
)
```

`woods_plot` *Woods' plot*

Description

Creates a Woods' plot that plots log2 fold change of peptides or precursors along the protein sequence. The peptides or precursors are located on the x-axis based on their start and end positions. The position on the y-axis displays the fold change. The vertical size (y-axis) of the box representing the peptides or precursors do not have any meaning.

Usage

```
woods_plot(
  data,
  fold_change,
  start_position,
  end_position,
  protein_length,
  coverage = NULL,
  protein_id,
  targets = "all",
```

```
  facet = TRUE,
  colouring = NULL,
  fold_change_cutoff = 1,
  highlight = NULL,
  export = FALSE,
  export_name = "woods_plots"
)
```

Arguments

data	a data frame that contains 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	a numeric column in the data data frame that contains log2 fold changes.
start_position	a numeric column in the data data frame that contains the start positions for each peptide or precursor.
end_position	a numeric column in the data data frame that contains the end positions for each peptide or precursor.
protein_length	a numeric column in the data data frame that contains the length of the protein.
coverage	optional, a numeric column in the data data frame that contains coverage in percent. Will appear in the title of the Woods' plot if provided.
protein_id	a character column in the data data frame that contains protein identifiers.
targets	a character vector that specifies the identifiers of the proteins (depending on protein_id) that should be plotted. This can also be "all" if plots for all proteins should be created. Default is "all".
facet	a logical value that indicates if plots should be summarised into facets of 20 plots. This is recommended for many plots. Default is facet = TRUE.
colouring	optional, a character or numeric (discrete or continous) column in the data frame containing information by which peptide or precursors should be coloured.
fold_change_cutoff	optional, a numeric value that specifies the log2 fold change cutoff used in the plot. The default value is 2.
highlight	optional, a logical column that specifies whether specific peptides or precursors should be highlighted with an asterisk.
export	a logical value that indicates 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. Default is export = FALSE.
export_name	a character vector that provides the name of the exported file if export = TRUE. Default is export_name = "woods_plots"

Value

A list containing Woods' plots is returned. Plotting peptide or precursor log2 fold changes along the protein sequence.

Examples

```
# Create example data
data <- data.frame(
  fold_change = c(2.3, 0.3, -0.4, -4, 1),
  pval = c(0.001, 0.7, 0.9, 0.003, 0.03),
  start = c(20, 30, 45, 90, 140),
  end = c(33, 40, 64, 100, 145),
  protein_length = c(rep(150, 5)),
  protein_id = c(rep("P1", 5))
)

# Plot Woods' plot
woods_plot(
  data = data,
  fold_change = fold_change,
  start_position = start,
  end_position = end,
  protein_length = protein_length,
  protein_id = protein_id,
  colouring = pval
)
```

Index

∗ **datasets**
  protti_colours, 76
  ptsi_pgk, 77
  rapamycin_10uM, 101
  rapamycin_dose_response, 102
  viridis_colours, 107

analyse_functional_network, 3
anova_protti, 5
assign_missingness, 7
assign_peptide_type, 9

barcode_plot, 10

calculate_aa_scores, 11
calculate_diff_abundance, 13
calculate_go_enrichment, 16
calculate_imputation, 19
calculate_kegg_enrichment, 20
calculate_protein_abundance, 22
calculate_sequence_coverage, 24
calculate_treatment_enrichment, 25
create_queue, 26
create_structure_contact_map, 30
create_synthetic_data, 33

drc_4p, 35
drc_4p_plot, 36

extract_metal_binders, 38

fetch_alphafold_prediction, 40
fetch_chebi, 42
fetch_eco, 43
fetch_go, 44
fetch_kegg, 45
fetch_metal_pdb, 45
fetch_mobidb, 48
fetch_pdb, 48
fetch_pdb_structure, 49
fetch_uniprot, 51
fetch_uniprot_proteome, 52
filter_cv, 53
find_all_subs, 54
find_chebis, 55
find_peptide, 55
find_peptide_in_structure, 56
fit_drc_4p, 58

impute, 62

map_peptides_on_structure, 64

normalise, 67

parallel_create_structure_contact_map, 68
parallel_fit_drc_4p, 71
peptide_profile_plot, 74
protti_colours, 76
ptsi_pgk, 77
pval_distribution_plot, 78

qc_charge_states, 79
qc_contaminants, 80
qc_cvs, 82
qc_data_completeness, 83
qc_ids, 85
qc_intensity_distribution, 86
qc_median_intensities, 88
qc_missed_cleavages, 89
qc_pca, 91
qc_peak_width, 92
qc_peptide_type, 94
qc_proteome_coverage, 96
qc_sample_correlation, 97
qc_sequence_coverage, 99

randomise_queue, 100
rapamycin_10uM, 101
rapamycin_dose_response, 102
read_protti, 103

replace_identified_by_x, 103

scale_protti, 104
sequence_coverage, *99*
split_metal_name, 105

try_query, 105
ttest_protti, 106

viridis_colours, 107
volcano_plot, 107

woods_plot, 110