Package: tidyproteomics (via r-universe)

September 5, 2024

Title An S3 data object and framework for common quantitative proteomic analyses

Version 1.8.2

Description Creates a simple, universal S3 data structure for the post analysis of mass spectrometry based quantitative proteomic data. In addition, this package collects, adapts and organizes several useful algorithms and methods used in typical post analysis workflows.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

VignetteBuilder knitr

Suggests utils, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports Biostrings, broom, cli, digest, doParallel, dplyr, e1071, eulerr, fgsea, forcats, ggplot2, ggrepel, glue, grDevices, limma, magrittr, methods, missForest, parallel, pheatmap, purrr, randomForest, RColorBrewer, readr, readxl, rlang, scales, stringr, tibble, tidyr, tidyselect, vroom, writexl

Depends R (>= 3.5.0)

LazyData true

URL `https://jeffsocal.github.io/tidyproteomics/`

Repository https://blaserlab.r-universe.dev

RemoteUrl https://github.com/jeffsocal/tidyproteomics

RemoteRef HEAD

RemoteSha fe0d14a6b5962aa8a07e35aa8752aa9e164d102a

Contents

`align_modification` *Align a modification to a peptide sequence*

Description

Align a modification to a peptide sequence

Usage

```
align_modification(peptide = NULL, modification = NULL)
```

Arguments

`peptide`	a character string representing a peptide sequence
`modification`	a character string representing a modification and location probability

Value

a tidyproteomics data-object

`align_peptide` *Align a peptide sequence to a protein sequence*

Description

Align a peptide sequence to a protein sequence

Usage

```
align_peptide(peptide = NULL, protein = NULL)
```

Arguments

`peptide`	a character string representing a peptide sequence
`protein`	a character string representing a protein sequence

Value

a tidyproteomics data-object

analysis_counts *A function for evaluating expression differences between two sample sets via the limma algorithm*

Description

A function for evaluating expression differences between two sample sets via the limma algorithm

Usage

```
analysis_counts(data = NULL, impute_max = 0.5)
```

Arguments

`data`	tidyproteomics data object
`impute_max`	a numeric representing the largest allowable imputation percentage

Value

a tibble

analyze_enrichments *Analysis tables and plots of expression values*

Description

`analyze_enrichments()` is a GGplot2 implementation for plotting the expression differences as foldchange ~ statistical significance. See also `plot_proportion()`. This function can take either a tidyproteomics data object or a table with the required headers.

Usage

```
analyze_enrichments(
  data = NULL,
  top_n = 50,
  significance_max = 0.05,
  enriched_up_color = "blue",
  enriched_down_color = "red",
  height = 6.5,
  width = 10
)
```

Arguments

`data`	a character defining the column name of the log2 foldchange values.
`top_n`	a numerical value defining the number of terms to display in the plot
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`enriched_up_color`	a color to assign the up enriched values
`enriched_down_color`	a color to assign the down enriched values
`width`	a numeric

Value

a tidyproteomics data object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   analyze_expressions(log2fc_min = 0.5, significance_column = "p_value")
```

`analyze_expressions` *Analysis tables and plots of expression values*

Description

`analyze_expressions()` is a GGplot2 implementation for plotting the expression differences as foldchange ~ statistical significance. See also `plot_proportion()`. This function can take either a tidyproteomics data object or a table with the required headers.

Usage

```
analyze_expressions(
  data = NULL,
  log2fc_min = 1,
  log2fc_column = "log2_foldchange",
  significance_max = 0.05,
  significance_column = "adj_p_value",
  labels_column = NULL,
  show_pannels = TRUE,
  show_lines = TRUE,
  show_fc_scale = TRUE,
  show_title = TRUE,
  show_pval_1 = TRUE,
  point_size = NULL,
  color_positive = "dodgerblue",
  color_negative = "firebrick1",
  height = 5,
  width = 8
)
```

Arguments

`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`labels_column`	a character defining the column name of the column for labeling.
`show_pannels`	a boolean for showing colored up/down expression panels.
`show_lines`	a boolean for showing threshold lines.
`show_fc_scale`	a boolean for showing the secondary foldchange scale.
`show_title`	input FALSE, TRUE for an auto-generated title or any charcter string.
`show_pval_1`	a boolean for showing expressions with pvalue == 1.
`point_size`	a character reference to a numerical value in the expression table
`color_positive`	a character defining the color for positive (up) expression.
`color_negative`	a character defining the color for negative (down) expression.
`height`	a numeric
`width`	a numeric

Value

a tidyproteomics data object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   analyze_expressions(log2fc_min = 0.5, significance_column = "p_value")
```

annotate *Main function for adding annotations to a tidyproteomics data-object*

Description

Main function for adding annotations to a tidyproteomics data-object

Usage

```
annotate(
  data = NULL,
  annotations = NULL,
  duplicates = c("replace", "merge", "leave")
)
```

Arguments

`data`	a tidyproteomics data list-object
`annotations`	a character string vector
`duplicates`	a character string, how to handle duplicate terms

Value

a tidyproteomics data list-object

as.data.frame.tidyproteomics *Helper function to convert the data-object into a tibble*

Description

`as.data.frame()` is a function that converts the tidyproteomics data object into a tibble. This tibble is in the long-format, such that a there is a single observation per line.

Usage

```
## S3 method for class 'tidyproteomics'
as.data.frame(data, shape = c("long", "wide"), values = NULL, drop = NULL)
```

Arguments

`data`	tidyproteomics data object
`shape`	the orientation of the quantitative data as either a single measure per row (**long**), or as multiple measures per protein/peptide (**wide**).
`values`	indicates the selected normalization to output. The default is that selected at the time of normalization.

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# convert the data-object to a data.frame
hela_proteins %>% as.data.frame() %>% as_tibble()

# select the wide format
hela_proteins %>% as.data.frame(shape = 'wide') %>% as_tibble()

# select the wide format & drop some columns
hela_proteins %>%
   as.data.frame(shape = 'wide',
           drop = c('description','wiki_pathway','reactome_pathway','biological_process')) %>%
   as_tibble()
```

`calc_enrichment` *Helper function to calculate term enrichment*

Description

Helper function to calculate term enrichment

Usage

```
calc_enrichment(data, x)
```

Arguments

`data`	tidyproteomics data table object
`x`	the annotation to compute enrichment for

Value

list of vectors

`center` *helper function for normalizing a quantitative table*

Description

helper function for normalizing a quantitative table

Usage

```
center(
  table,
  group_by = c("identifier"),
  values = "abundance",
  method = c("median", "mean", "geomean", "sum")
)
```

Arguments

`table`	a tibble
`group_by`	character vector
`values`	character string
`method`	character string

Value

a tibble

`check_data` *Check the integrity of a tidyproteomics data object*

Description

`check_data()` is a helper function that checks the structure and contents of a tidyproteomics data object

Usage

```
check_data(data = NULL)
```

Arguments

`data`	tidyproteomics data object

Value

silent on success, an abort message on fail

check_pairs *Helper function for iterative expression analysis*

Description

Helper function for iterative expression analysis

Usage

```
check_pairs(pairs = NULL, sample_names = NULL)
```

Arguments

`pairs`	the list of vector doublets
`data`	tidyproteomics data object

Value

list of vectors

check_table *Check the integrity of a tidyproteomics quantitative tibble*

Description

`check_table()` is a helper function that checks the structure and contents of a tidyproteomics quantitative tibble

Usage

```
check_table(table = NULL)
```

Arguments

`table`	a tibble

Value

silent on success, an abort message on fail

codify — *Build a tidyproteomics data object*

Description

`data_codify()` is a helper function

Usage

```
codify(table = NULL, identifier = NULL, annotations = NULL)
```

Arguments

`table`	tidyproteomics data object
`identifier`	a character vector
`annotations`	a character vector

Value

tidyproteomics data object

collapse — *Convert peptide quantitative data into protein quantitative data*

Description

`collapse()` produces a protein based tidyproteomics data-object from a peptide based tidyproteomics data-object.

Usage

```
collapse(
  data = NULL,
  collapse_to = "protein",
 assign_by = c("all-possible", "razor-local", "razor-global", "non-homologous"),
  top_n = Inf,
  split_abundance = FALSE,
  fasta_path = NULL,
  .verbose = TRUE,
  .function = fsum
)
```

Arguments

`data`	a tidyproteomics data-object
`collapse_to`	a character string representing the final aggregation point. Conventionally this is the protein name or id, however, if a gene_name or any other term exists in the annotations table of the data-object, peptides can be aggregated to that.
`assign_by`	the method to by which to combine peptides into proteins; **all-possible** allows peptide's quantitative value to be included in all assigned proteins, **razor-local** (razor peptides are shared between proteins, a peptide which could belong to different proteins is assigned to the protein that has the highest likelihood to be actually present in the sample, so the shared peptide can only contribute to the identification score of the protein group which has the highest probability of being in the sample), in this case assignment goes to the protein of highest probability only within a sample class, such that peptides from another sample group which change the protein of highest probability are not accounted for in this scheme. **razor-global** determines protein of highest probability using all available peptides in the data set, **non-homologous** only utilizes the abundance values from peptides that have a single unique identity.
`top_n`	a numeric to indicate the N number of peptides summed account for the protein quantitative value, this assumes that peptides have been summed across charge states
`split_abundance`	(experimental) a boolean to indicate if abundances for razor peptides should be split according to protein prevalence, or the proportion of total abundance between all proteins that share a particular peptide.
`fasta_path`	if supplied, it will be used to fill in annotation values such as description, protein_name and gene_name
`.verbose`	a boolean
`.function`	an assignable protein abundance summary function, fsum, fmean, fgeomean and fmedian have constructed as NAs must be removed. The default is fsum() `fsum <- function(x){base::sum(x, na.rm = TRUE)}`, where x is the vector of peptide abundances assigned to that protein by the `assign_by` method. Note - peptides that have a 0 or NA quantitative value are still used to determine razor assignments, as that sequence was observed, quantitative values are just missing.

Value

a tidyproteomics data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
# data <- hela_peptides %>% collapse()
# data %>% summary("sample")
```

compute_compexp *Helper function to analysis between two expression tests*

Description

Helper function to analysis between two expression tests

Usage

```
compute_compexp(
  table_a = NULL,
  table_b = NULL,
  log2fc_min = 2,
  log2fc_column = "log2_foldchange",
  significance_max = 0.05,
  significance_column = "adj_p_value",
  labels_column = "protein"
)
```

Arguments

`table_a`	a tibble
`table_b`	a tibble
`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`labels_column`	a character defining the column name of the column for labeling.

Value

a list

data_import *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
data_import(file_names = NULL, platform = NULL, analyte = NULL, path = NULL)
```

Arguments

`file_names`	a character vector of file paths
`platform`	a character string
`analyte`	a character string
`path`	a character string

Value

a tidyproteomics list data-object

`down_select` *Helper function to subset a data frame*

Description

Helper function to subset a data frame

Usage

```
down_select(table = NULL, tidyproteomics_quo = NULL)
```

Arguments

`table`	a tibble
`tidyproteomics_quo`	a character vector

Value

a tibble

`enrichment` *Compute protein enrichment*

Description

`enrichment()` is an analysis function that computes the protein summary statistics for a given tidyproteomics data object.

Usage

```
enrichment(
  data = NULL,
  ...,
  .pairs = NULL,
  .terms = NULL,
  .method = c("gsea", "wilcoxon", "fishers_exact"),
  .score_type = c("std", "pos", "neg"),
  .log2fc_min = 0,
  .significance_min = 0.05,
  .cpu_cores = 1
)
```

Arguments

`data`	tidyproteomics data object
`...`	two sample comparison e.g. experimental/control
`.pairs`	a list of vectors each containing two named sample groups
`.terms`	a character string referencing "term(s)" in the annotations table
`.method`	a character string
`.score_type`	a character string. From the fgsea manual: "This parameter defines the GSEA score type. Possible options are ("std", "pos", "neg"). By default ("std") the enrichment score is computed as in the original GSEA. The "pos" and "neg" score types are intended to be used for one-tailed tests (i.e. when one is interested only in positive ("pos") or negateive ("neg") enrichment)."
`.log2fc_min`	used only for Fisher's Exact Test, a numeric defining the minimum log2 foldchange to consider as "enriched"
`.cpu_cores`	the number of threads used to speed the calculation
`.significance_max`	used only for Fisher's Exact Test, a numeric defining the maximum statistical significance to consider as "enriched"

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# using the default GSEA method
hela_proteins %>%
   expression(knockdown/control) %>%
   enrichment(knockdown/control, .terms = "biological_process") %>%
  export_analysis(knockdown/control, .analysis = "enrichment", .term = "biological_process")
```

```
# using a Wilcoxon Rank Sum method
hela_proteins %>%
   expression(knockdown/control) %>%
   enrichment(knockdown/control, .terms = "biological_process", .method = "wilcoxon") %>%
  export_analysis(knockdown/control, .analysis = "enrichment", .term = "biological_process")

# using the .pairs argument when multiple comparisons are needed
comps <- list(c("control","knockdown"),
            c("knockdown","control"))

hela_proteins %>%
   expression(.pairs = comps) %>%
   enrichment(.pairs = comps, .terms = c("biological_process", "molecular_function")
```

enrichment_fishersexact

A function for evaluating term enrichment via Fischer's Exact method

Description

A function for evaluating term enrichment via Fischer's Exact method

Usage

```
enrichment_fishersexact(
  data_expression = NULL,
  data = NULL,
  term_group = NULL,
  log2fc_min = 0,
  significance_min = 0.05,
  cpu_cores = 1,
  ...
)
```

Arguments

`data_expression`	a tibble from and two sample expression difference analysis
`data`	tidyproteomics data object
`term_group`	a character string referencing "term" in the annotations table
`log2fc_min`	a numeric defining the minimum log2 foldchange to consider as "enriched"
`cpu_cores`	the number of threads used to speed the calculation
`...`	pass through arguments
`significance_max`	a numeric defining the maximum statistical significance to consider as "enriched"

Value

a tibble

`enrichment_gsea` *A function for evaluating term enrichment via GSEA*

Description

A function for evaluating term enrichment via GSEA

Usage

```
enrichment_gsea(
  data_expression = NULL,
  data = NULL,
  term_group = NULL,
  score_type = c("std", "pos", "neg"),
  cpu_cores = 1
)
```

Arguments

`data_expression`	a tibble from and two sample expression difference analysis
`data`	tidyproteomics data object
`term_group`	a character string referencing "term" in the annotations table
`score_type`	a character string used in the fgsea package
`cpu_cores`	the number of threads used to speed the calculation

Value

a tibble

`enrichment_wilcoxon` *A function for evaluating term enrichment via Wilcoxon Rank Sum*

Description

A function for evaluating term enrichment via Wilcoxon Rank Sum

Usage

```
enrichment_wilcoxon(
  data_expression = NULL,
  data = NULL,
  term_group = NULL,
  cpu_cores = 1,
  ...
)
```

Arguments

`data_expression`	a tibble from and two sample expression difference analysis
`data`	tidyproteomics data object
`term_group`	a character string referencing "term" in the annotations table
`cpu_cores`	the number of threads used to speed the calculation
`...`	pass through arguments

Value

a tibble

`experimental` *Returns the data experimental set up*

Description

`experimental()` returns the transformative operations performed on the data.

Usage

```
experimental(data = NULL, destination = c("print", "save"))
```

Arguments

`data`	tidyproteomics data object
`destination`	a character string

Value

a character

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
#\dontrun{
hela_proteins <- path_to_package_data("p97KD_HCT116") %>%
   import("ProteomeDiscoverer", "proteins") %>%
   reassign(sample == "ctl", .replace = "control") %>%
   reassign(sample == "p97", .replace = "knockdown") %>%
   impute() %>%
   normalize(.method = c("linear","loess"))
}
hela_proteins %>% experimental()
```

experimental_groups *Main function for adding sample groups*

Description

Main function for adding sample groups

Usage

```
experimental_groups(data = NULL, sample_groups = NULL)
```

Arguments

data	a tidyproteomics data list-object
sample_groups	a character string vector equal to the experimental row length

Value

a tidyproteomics data list-object

export_analysis *Export the quantitative data from an tidyproteomics data-object*

Description

`export_analysis()` returns the main quantitative data object as a tibble with *identifier* as the designation for the measured observation.

Usage

```
export_analysis(
  data = NULL,
  ...,
  .analysis = NULL,
  .term = NULL,
  .append = NULL,
  .file_name = NULL
)
```

Arguments

`data`	tidyproteomics data object
`...`	two sample comparison e.g. experimental/control
`.analysis`	a character string for the specific analysis to export. For example, the base analysis 'counts' always exists, it is the base analysis supporting plot_counts(). The other analysis are 'expression' and 'enrichment', which are only available when those analyses have been performed.
`.term`	a character string of the term from an enrichment analysis. Use the show_annotations() function to list the available terms.
`.append`	a character string of the term to append to the output. Use the show_annotations() function to list the available terms.
`.file_name`	a character string for file to write to, format implied from string ('.rds', '.xlsx', '.csv', '.tsv')

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   export_analysis(knockdown/control,
                   .analysis = "expression")

hela_proteins %>%
   export_analysis(.analysis = "counts")
```

`export_compexp` *Comparative analysis between two expression tests*

Description

`export_compexp()` returns a table of the comparison in expression differences between two methods or two sets of groups. For example, one could run an expression difference for two different conditions (A and B) prodived the experiment contained 3 samples condition A, condition B and WT, then compare those results. The proteins showing up in the intersection indicate common targets for condition A and B.

```
expdiff_a <- protein_data %>%
   expression(experiment = "condition_a", control = "wt")

expdiff_b <- protein_data %>%
   expression(experiment = "condition_b", control = "wt")

export_compexp(expdiff_a, expdiff_b, export = "intersect")
```

Usage

```
export_compexp(
  table_a = NULL,
  table_b = NULL,
  log2fc_min = 2,
  log2fc_column = "log2_foldchange",
  significance_max = 0.05,
  significance_column = "adj_p_value",
  labels_column = "protein",
  export = c("all", "a_only", "b_only", "intersect")
)
```

Arguments

`table_a`	a tibble
`table_b`	a tibble
`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`labels_column`	a character defining the column name of the column for labeling.
`export`	a character string for the significance data to return

Value

a tibble

`export_config` *Helper function to export the config file to current project directory*

Description

Helper function to export the config file to current project directory

Usage

```
export_config(platform = NULL, analyte = c("proteins", "peptides"))
```

Arguments

`platform`	the source of the data (ProteomeDiscoverer, MaxQuant)
`analyte`	the omics analyte (proteins, peptides)

Value

success or fail

Examples

```
library(tidyproteomics)
#\dontrun{
export_config("mzTab", 'peptides')
}
```

`export_quant` *Export the quantitative data from an tidyproteomics data-object*

Description

`export_quant()` returns the main quantitative data object as a tibble with *identifier* as the designation for the measured observation.

Usage

```
export_quant(
  data = NULL,
  file_name = NULL,
  raw_data = TRUE,
  normalized = FALSE,
  scaled = c("none", "between", "proportion")
)
```

Arguments

`data`	tidyproteomics data object
`file_name`	character string vector
`raw_data`	a boolean
`normalized`	a boolean
`scaled`	a boolean

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   normalize(.method = "loess") %>%
   export_quant(file_name = "hela_quant_data.xlsx", normalized = "loess")
```

`expression` *Summarize the data*

Description

`expression()` is an analysis function that computes the protein summary statistics for a given tidyproteomics data object.

Usage

```
expression(
  data = NULL,
  ...,
  .pairs = NULL,
  .method = stats::t.test,
  .p.adjust = "BH"
)
```

Arguments

`data`	tidyproteomics data object
`...`	two sample comparison e.g. experimental/control
`.method`	a two-distribution test function returning a p_value for the null hypothesis. Example functions include t.test, wilcox.test, stats::ks.test, additionally, the string *"limma"* can be used to select from the limma package to compute an empirical Bayesian estimation which performs better with non-linear distributions and uneven replicate balance between samples.

`.p.adjust`	a stats::p.adjust string for multiple test correction, default is 'BH' (Benjamini & Hochberg, 1995)

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# simple t.test expression analysis
hela_proteins %>%
   expression(knockdown/control) %>%
   export_analysis(knockdown/control, .analysis = "expression")

# a wilcox.test expression analysis
hela_proteins %>%
   expression(knockdown/control, .method = stats::wilcox.test) %>%
   export_analysis(knockdown/control, .analysis = "expression")

# a one-tailed wilcox.test expression analysis
wilcoxon_less <- function(x, y) {
   stats::wilcox.test(x, y, alternative = "less")
}
hela_proteins <- hela_proteins %>%
   expression(knockdown/control, .method = stats::wilcox.test)

hela_proteins %>% export_analysis(knockdown/control, .analysis = "expression")

# Note: the userdefined function is preserved in the operations tracking
hela_proteins %>% operations()

# limma expression analysis
hela_proteins %>%
   expression(knockdown/control, .method = "limma") %>%
   export_analysis(knockdown/control, .analysis = "expression")

# using the .pairs argument when multiple comparisons are needed
comps <- list(c("control","knockdown"),
            c("knockdown","control"))

hela_proteins %>%
   expression(.pairs = comps)
```

`expression_limma` *Calculate expression differences between two-samples*

Description

`expression_limma()` is a function for evaluating expression differences between two sample sets via the limma algorithm

Usage

```
expression_limma(data = NULL, experiment = NULL, control = NULL)
```

Arguments

`data`	tidyproteomics data object
`experiment`	a character string representing the experimental sample set
`control`	a character string representing the control sample set

Value

a tibble

`expression_test` *A function for evaluating expression differences between two sample sets via the limma algorithm*

Description

A function for evaluating expression differences between two sample sets via the limma algorithm

Usage

```
expression_test(
  data = NULL,
  experiment = NULL,
  control = NULL,
  .method = stats::t.test,
  ...,
  .p.adjust = "BH"
)
```

Arguments

`data`	tidyproteomics data object
`experiment`	a character string representing the experimental sample set
`control`	a character string representing the control sample set
`.method`	a two-distribution test function returning a p_value for the null hypothesis. Default is t.test. Example functions include t.test, wilcox.test, stats::ks.test ...
`...`	pass through arguments
`.p.adjust`	a stats::p.adjust string for multiple test correction

Value

a tibble

---

extract — *Main function for extracting quantitative data from a tidyproteomics data-object*

Description

Main function for extracting quantitative data from a tidyproteomics data-object

Usage

```
extract(data = NULL, values = NULL, na.rm = FALSE)
```

Arguments

`data`	tidyproteomics data object
`values`	character string vector
`na.rm`	a boolean

Value

a tibble

---

fasta_digest — *Proteolytic digest a parsed fasta list*

Description

`fasta_digest()` Generates peptide sequences based on *enzyme* and *partial* inputs. Only works with the "list" output of the `parse()` function

Usage

```
fasta_digest(protein = NULL, ...)
```

Arguments

`protein`	as character string
`...`	parameters for `peptides()`

Value

a list

Examples

```
#\dontrun{
proteins <- fasta_parse("~/Local/data/fasta/ecoli_UniProt.fasta")
proteins <- fasta_digest(proteins, enzyme = "[K]", partial = 2)
}
```

`fasta_extract` *Get the string defined by the regex*

Description

`fasta_extract()` get the current string based on regex

Usage

```
fasta_extract(string = NULL, regex = NULL)
```

Arguments

`string`	a character
`regex`	a list

Value

a list

`fasta_parse` *The main function for parsing a fasta file*

Description

`fasta_parse()` get the current regex

Usage

```
fasta_parse(fasta_path = NULL, patterns = NULL, as = c("list", "data.frame"))
```

Arguments

`fasta_path`	a character string of the path to the fasta formatted file
`patterns`	a list, if not provided the default from `regex()` will be used. *Note*: the first element in the regex list will define the list reference name, such that with the list output, each protein can be accessed with that designation. *Note*: if the patterns list is missing an explicit "sequence" element, no sequence will be returned. This might be beneficial if only a few meta elements are sought.
`as`	a character designating the output format

Value

a list

Examples

```
#\dontrun{
proteins <- fasta_parse("~/Local/data/fasta/ecoli_UniProt.fasta")

# using a custom supplied regex list
proteins <- fasta_parse(fasta_path = "~/Local/data/fasta/ecoli_UniProt.fasta",
                        pattern = list(
                        "accession" = "sp\\|[A-Z]",
                        "gene_name" = "(?<=GN\\=).*?(?=\\s..\\=)"
                  ))
}
```

`fasta_peptides` *Proteolytic digest a sequence*

Description

`fasta_peptides()` Generates peptide sequences based on `enzyme` and `partial` inputs.

Usage

```
fasta_peptides(
  sequence = NULL,
  enzyme = "[KR]",
  partial = 0:3,
  length = c(6, 30)
)
```

Arguments

`sequence`	as character string
`enzyme`	a character string regular expression use to proteolytically digest the sequence. • `[KR]` ... trypsin • `[KR](?!P)` ... trypsin not at P • `[R](?!P)` ... arg-c • `[K](?!P)` ... lys-c • `[FYWL](?!P)` ... chymotrypsin • `[BD]` ... asp-n • `[D]` ... formic acid • `[FL]` ... pepsin-a
`partial`	a numeric representing the number of incomplete enzymatic sites (mis-clevage).
`length`	as numeric vactor representing the minimum and maximum sequence lengths.

Value

a vector

Examples

```
#\dontrun{
sequence <- "SAMERSMALLKPSAMPLERSEQUENCE"
tidyproteomics:::fasta_peptides(sequence)

tidyproteomics:::fasta_peptides(sequence, enzyme = "[L]", partial = 2, length = c(1,12))

}
```

`fasta_regex` *Get/Set the FASTA meta data regex*

Description

`fasta_regex()` gets and sets the current regex patters to assist the `parse()` function. This simply provides the structure needed to parse the fasta file, a custom list can also be supplied. To set elements in the `regex()` function, simply provide a list with complementary names to over-write the current list.

Usage

```
fasta_regex(params = NULL)
```

Arguments

`params`	as list

Value

a list

Examples

```
#\dontrun{
fasta_regex(list("accession" = "sp\\|[A-Z]"))
}
```

`fgeomean` *Calculates the geometric mean of a numeric vector with NAs removed*

Description

Calculates the geometric mean of a numeric vector with NAs removed

Usage

```
fgeomean(x)
```

Arguments

`x`	a numeric vector

Value

a numeric

Examples

```
library(tidyproteomics)
fgeomean(c(1,2,5,6,8,NA,NA))
```

`fmean` *Calculates the mean of a numeric vector with NAs removed*

Description

Calculates the mean of a numeric vector with NAs removed

Usage

```
fmean(x)
```

Arguments

`x`	a numeric vector

Value

a numeric

Examples

```
library(tidyproteomics)
fmean(c(1,2,5,6,8,NA,NA))
```

fmedian — *Calculates the median of a numeric vector with NAs removed*

Description

Calculates the median of a numeric vector with NAs removed

Usage

```
fmedian(x)
```

Arguments

`x`	a numeric vector

Value

a numeric

Examples

```
library(tidyproteomics)
fmedian(c(1,2,5,6,8,NA,NA))
```

fmin — *Calculates the minimum of a numeric vector with NAs removed*

Description

Calculates the minimum of a numeric vector with NAs removed

Usage

```
fmin(x)
```

Arguments

`x`	a numeric vector

Value

a numeric

Examples

```
library(tidyproteomics)
fmin(c(1,2,5,6,8,NA,NA))
```

`fsum` *Calculates the sum of a numeric vector with NAs removed*

Description

Calculates the sum of a numeric vector with NAs removed

Usage

```
fsum(x)
```

Arguments

`x`	a numeric vector

Value

a numeric

Examples

```
library(tidyproteomics)
fsum(c(1,2,5,6,8,NA,NA))
```

`get_accountings` *Helper function to get all accounting terms*

Description

Helper function to get all accounting terms

Usage

```
get_accountings(data = NULL)
```

Arguments

`data`	tidyproteomics data object

Value

a vector

`get_annotations` *Helper function to get all annotations for a given term*

Description

Helper function to get all annotations for a given term

Usage

```
get_annotations(data = NULL, term = NULL)
```

Arguments

`data`	tidyproteomics data object
`term`	a character string

Value

a vector

`get_annotation_terms` *Helper function to get available terms*

Description

Helper function to get available terms

Usage

```
get_annotation_terms(data)
```

Arguments

`data`	tidyproteomics data object

Value

a vector

`get_quant_names` *Get the quantitative value names*

Description

`get_quant_names()` is a helper function that returns the names for all of the normalized quantitative values, such as *raw*, *linear*, *loess*

Usage

```
get_quant_names(data)
```

Arguments

`data`	a tidyproteomics data-object

Value

a character vector

Examples

```
library(tidyproteomics)
get_quant_names(hela_proteins)
```

`get_sample_names` *Helper function to get all sample names*

Description

Helper function to get all sample names

Usage

```
get_sample_names(data = NULL)
```

Arguments

`data`	tidyproteomics data object

Value

a vector

`get_segment` *Helper function to get available terms*

Description

Helper function to get available terms

Usage

```
get_segment(data = NULL, variable = NULL, .verbose = TRUE)
```

Arguments

`data`	tidyproteomics data object
`variable`	a character string
`.verbose`	a boolean

Value

a character

`get_unique_variables` *Helper function to get all sample names*

Description

Helper function to get all sample names

Usage

```
get_unique_variables(data = NULL, variable = NULL)
```

Arguments

`data`	tidyproteomics data object
`variable`	a string character

Value

a vector

get_variables *Helper function to get available terms*

Description

Helper function to get available terms

Usage

```
get_variables(
  data = NULL,
  segment = c("experiments", "quantitative", "annotations", "accounting")
)
```

Arguments

data	tidyproteomics data object
segment	a character string

Value

a vector

hash_vector *Create a crc32 hash on a vector*

Description

`hash_vector()` is a helper function that returns a crc32 hash on a vector

Usage

```
hash_vector(x)
```

Arguments

x	a vector

Value

a hash of x

hdf *Helper function to take the head of a tibble and display as a data.frame*

Description

Helper function to take the head of a tibble and display as a data.frame

Usage

```
hdf(x, n = 5)
```

Arguments

`x`	a tibble
`n`	display up to the nth row

Value

a data frame

Examples

```
library(tidyproteomics)
x <- tibble::tibble(a = 1:10, b = 11:20)
hdf(x)
hdf(x, n = 3)
```

hela_peptides *A sample tidyproteomics data object*

Description

A dataset containing the quantitative peptide data for ten proteins from 2 samples with 3 replicates each

Usage

```
hela_peptides
```

Format

A list collection of character values and tibbles:

quantitative tibble, protein quantitative data

annotation tibble, protein annotation data ...

`hela_proteins` *A sample tidyproteomics data object*

Description

A dataset containing the quantitative protein data for thousands of proteins from 2 samples with 3 replicates each

Usage

```
hela_proteins
```

Format

A list collection of character values and tibbles:

quantitative tibble, protein quantitative data

annotation tibble, protein annotation data ...

`import` *Main function for importing data*

Description

`import()` reads files from various platforms into the tidyproteomics data object – see also the documentation `vignette("importing")` and `vignette("workflow-importing")`

Usage

```
import(files = NULL, platform = NULL, analyte = NULL, path = NULL)
```

Arguments

`files`	a character vector of file paths
`platform`	the source of the data (ProteomeDiscoverer, MaxQuant, etc.)
`analyte`	the omics analyte (proteins, peptides)
`path`	a character string pointing to the local configuration file (directory/file.tsv)

Value

a tidyproteomics list data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins <- path_to_package_data("p97KD_HCT116") %>%
   import("ProteomeDiscoverer", "proteins")
hela_proteins %>% summary("sample")
```

---

import_extract   *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_extract(tbl_data = NULL, tbl_config = NULL, remove = FALSE)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values
`remove`	as boolean to determine if the extracted column name should change or copy to a new, retaining the old

Value

a tibble

---

import_mbr   *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_mbr(tbl_data = NULL, tbl_config = NULL)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values

Value

a tibble

`import_remove` *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_remove(tbl_data = NULL, tbl_config = NULL)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values

Value

a tibble

`import_rename` *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_rename(tbl_data = NULL, tbl_config = NULL)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values

Value

a tibble

`import_split` *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_split(tbl_data = NULL, tbl_config = NULL)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values

Value

a tibble

`import_validate` *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
import_validate(tbl_data = NULL, tbl_config = NULL)
```

Arguments

`tbl_data`	a table of imported data
`tbl_config`	a table of config values

Value

a tibble

impute *Main method for imputing missing values*

Description

Main method for imputing missing values

Usage

```
impute(
  data = NULL,
  .function = base::min,
  method = c("row", "column", "matrix"),
  group_by_sample = FALSE,
  cores = 2
)
```

Arguments

`data`	a tidyproteomics list data-object
`.function`	summary statistic function. Default is base::min, examples of other functions include min, max, mean, sum. Note, NAs will be be removed in the function call.
`method`	a character string to indicate the imputation method (row, column, matrix). Consider a data matrix of peptide/protein "rows" and dataset "columns". A 'row' functions by imputing values between samples looking at the values for a given peptide/protein, while the 'column' method imputes within a dataset of values. The function 'randomforest' imputes using data from all rows and columns, or the "matrix", without bias toward sample groups. If given a bias for sample groups, expression differences would also bias sample groups. If it is the case that sample groups should be biased (such as gene deletion), then it is suggested to impute using min function and the 'within' method.
`group_by_sample`	a boolean to indicate that the data should be grouped by sample name to bias the imputation to within that sample.
`cores`	the number of threads used to speed the calculation

Value

a tidyproteomics list data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>% summary("sample")
hela_proteins %>% impute(.function = stats::median) %>% summary("sample")

hela_proteins %>% impute(.function = impute.randomforest) %>% summary("sample")
```

impute.randomforest *Imputes missing values based on the missForest function*

Description

Imputes missing values based on the missForest function

Usage

```
impute.randomforest(matrix = NULL, cores = 2)
```

Arguments

matrix	a matrix with some NAs
cores	the number of threads used to speed the calculation

Value

a matrix with imputed values

impute_ratio *Helper function for calculating imputation stats*

Description

Helper function for calculating imputation stats

Usage

```
impute_ratio(x)
```

Arguments

x	a tibble

Value

list of vectors

intersection *Create a data subset*

Description

`intersection()` is a specalized function for sub-setting quantitative data from a tidyproteomics data-object based data overlapping between sample groups.

Usage

```
intersection(data = NULL, .include = NULL, .exclude = NULL)
```

Arguments

`data`	tidyproteomics data object
`.include`	when exporting the "intersection" this is the set of proteins contained within the intersection of these samples
`.exclude`	when exporting the "intersection" this is the set of proteins found in these samples to exclude

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# creates a subset of just the proteins found in 'control'
hela_proteins %>%
   subset(imputed == 0) %>%
   intersection(.include = c('control'), .exclude = c('knockdown'))
```

intersect_venn *Helper function extracting a subset of proteins*

Description

Helper function extracting a subset of proteins

Usage

```
intersect_venn(data = NULL, include = NULL, exclude = NULL)
```

Arguments

`data`	the tidyproteomics data object
`include`	the set of proteins contained within the intersection of these samples
`exclude`	the set of proteins found in these samples to exclude

Value

a character string

`invlog2` *Inverse Log 2*

Description

Inverse Log 2

Usage

```
invlog2(x)
```

Arguments

`x`	Numeric value to calculate inverse log2

Value

A numeric

`list_venn` *Helper function Venn and Euler plots*

Description

Helper function Venn and Euler plots

Usage

```
list_venn(data = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`...`	pass through arguments

Value

list of vectors

`load_local` *Load project specific data*

Description

`load_local()` is a simple function that loads the current project tidyproteomics data object

Usage

```
load_local(analyte = c("peptides", "proteins"))
```

Arguments

`analyte`	a character string

Value

an tidyproteomics data object

Examples

```
library(tidyproteomics)
# hela_proteins <- load_omics(analyte = "proteins")
```

`match_vect` *match a named vector to string vector*

Description

match a named vector to string vector

Usage

```
match_vect(un_vec, n_vec)
```

Arguments

`un_vec`	an un-named vector
`n_vec`	a named vector

Value

a named vector

`meld` *Meld a tidyproteomics data object into a single table*

Description

`data_meld()` is a helper function

Usage

```
meld(data = NULL, single_quant_source = FALSE)
```

Arguments

`data`	tidyproteomics data object
`single_quant_source`	a boolean to indicate if only a single quantitative value should be reported

Value

a tibble

`merge` *Merge multiple tidyproteomics data-objects*

Description

`merge()` returns a single tidyproteomics data object from multiple.

Usage

```
merge(data_list = NULL, quantitative_source = c("raw", "selected", "all"))
```

Arguments

`data_list`	a list of tidyproteomics data objects
`quantitative_source`	a character string indicating which quantitative value to merge on. If `selected` is chosen then each dataset's specific normalization will be used and renamed to 'abundance_selected'. If `all` is chosen, then the possibility exists that some normalization values will fillin with NAs.

Value

a tidyproteomics data object

`merge_quantitative` *Helper function merging normalized data back into the main data-object*

Description

Helper function merging normalized data back into the main data-object

Usage

```
merge_quantitative(data = NULL, data_quant = NULL, values = "raw")
```

Arguments

`data`	tidyproteomics data subset tibble
`data_quant`	tidyproteomics data subset tibble
`values`	character string vector

Value

a tibble

`munge_identifier` *Main function for munging peptide data from an extracted tidyproteomics data-object*

Description

Main function for munging peptide data from an extracted tidyproteomics data-object

Usage

```
munge_identifier(
  data,
  munge = c("combine", "separate"),
  identifiers = c("protein", "peptide", "modifications")
)
```

Arguments

`data`	tidyproteomics data object
`munge`	character string vector (combine | separate)
`identifiers`	a character vector of the identifiers

Value

a tibble

`normalize` *Main function for normalizing quantitative data in a tidyproteomics data-object*

Description

`normalize()` Main function for normalizing quantitative data from a tidyproteomics data-object. This is a *passthrough* function as it returns the original tidyproteomics data-object with an additional quantitative column labeled with the normalization method(s) used.

This function can accommodate multiple normalization methods in a single pass, and it is useful for examining normalization effects on data. Often it is adventitious to select a optimal normalization method based on performance.

Usage

```
normalize(
  data,
  ...,
 .method = c("scaled", "median", "linear", "limma", "loess", "svm", "randomforest"),
  .cores = 1
)
```

Arguments

`data`	tidyproteomics data object
`...`	use a subset of the data for normalization see `subset()`. This is useful when normalizing against a spike-in set of proteins
`.method`	character vector of normalization to use
`.cores`	number of CPU cores to use for multi-threading

Value

a tidyproteomics data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
     normalize(.method = c("scaled", "median")) %>%
     summary("sample")

# normalize between samples according to a subset, then apply to all values
#   this would be recommended with a pull-down experiment wherein a conserved
#   protein complex acts as the majority content and individual inter-actors
#   are of quantitative differentiation
hela_proteins %>%
     normalize(!description %like% "Ribosome", .method = c("scaled", "median")) %>%
     summary("sample")
```

normalize_limma *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_limma(data = NULL)
```

Arguments

data	tidyproteomics data object

Value

a tibble

normalize_linear *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_linear(data = NULL, data_centered = NULL)
```

Arguments

data	tidyproteomics list data-object
data_centered	a tibble of centered values used for normalization

Value

a tibble

`normalize_loess` *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_loess(data = NULL, data_centered = NULL)
```

Arguments

`data`	tidyproteomics list data-object
`data_centered`	a tibble of centered values used for normalization

Value

a tibble

`normalize_median` *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_median(data = NULL, data_centered = NULL)
```

Arguments

`data`	tidyproteomics list data-object
`data_centered`	a tibble of centered values used for normalization

Value

a tibble

`normalize_randomforest` *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_randomforest(data = NULL, data_centered = NULL, .cores = 1)
```

Arguments

`data`	tidyproteomics list data-object
`data_centered`	a tibble of centered values used for normalization
`.cores`	number of CPU cores to use for multi-threading

Value

a tibble

`normalize_scaled` *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_scaled(data = NULL, data_centered = NULL)
```

Arguments

`data`	tidyproteomics list data-object
`data_centered`	a tibble of centered values used for normalization

Value

a tibble

`normalize_svm` *Normalization function for a tidyproteomics data-object*

Description

Normalization function for a tidyproteomics data-object

Usage

```
normalize_svm(data = NULL, data_centered = NULL, .cores = 1)
```

Arguments

`data`	tidyproteomics list data-object
`data_centered`	a tibble of centered values used for normalization
`.cores`	number of CPU cores to use for multi-threading

Value

a tibble

`operations` *Returns the data transformations*

Description

`operations()` returns the transformative operations performed on the data.

Usage

```
operations(data = NULL, destination = c("print", "save"))
```

Arguments

`data`	tidyproteomics data object
`destination`	a character string

Value

a character

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
#\dontrun{
hela_proteins <- path_to_package_data("p97KD_HCT116") %>%
   import("ProteomeDiscoverer", "proteins") %>%
   reassign(sample == "ctl", .replace = "control") %>%
   reassign(sample == "p97", .replace = "knockdown") %>%
   impute() %>%
   normalize(.method = c("linear","loess"))
}
hela_proteins %>% operations()
```

`path_to_package_data` *Helper function for displaying path to data*

Description

Helper function for displaying path to data

Usage

```
path_to_package_data(item = c("proteins", "peptides", "fasta"))
```

Arguments

`item`	a character string

Value

print the table to console

`plot.tidyproteomics` *Tidy-Quant data object plot definition*

Description

Tidy-Quant data object plot definition

Usage

```
## S3 method for class 'tidyproteomics'
plot(x, ...)
```

Arguments

`x`	tidyproteomics data object
`...`	unused legacy

Value

print object summary

`plot_compexp` *Comparative analysis between two expression tests*

Description

`plot_compexp()` is a GGplot2 implementation for plotting the comparison in expression differences between two methods or two sets of groups. For example, one could run an expression difference for two different conditions (A and B) prodived the experiment contained 3 samples condition A, condition B and WT, then compare those results. The proteins showing up in the intersection (purple) indicate common targets for condition A and B.

```
expdiff_a <- protein_data %>%
   expression(experiment = "condition_a", control = "wt")

expdiff_b <- protein_data %>%
   expression(experiment = "condition_b", control = "wt")

plot_compexp(expdiff_a, expdiff_b)
```

Usage

```
plot_compexp(
  table_a = NULL,
  table_b = NULL,
  log2fc_min = 2,
  log2fc_column = "log2_foldchange",
  significance_max = 0.05,
  significance_column = "adj_p_value",
  labels_column = "protein",
  point_size = NULL,
  show_lines = TRUE,
  color_a = "dodgerblue",
  color_b = "firebrick1",
  color_u = "purple"
)
```

Arguments

`table_a`	a tibble
`table_b`	a tibble
`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`labels_column`	a character defining the column name of the column for labeling.
`point_size`	a numeric for changing the point size.
`show_lines`	a boolean for showing threshold lines.
`color_a`	a character defining the color for table_a expression.
`color_b`	a character defining the color for table_b expression.
`color_u`	a character defining the color for the union between both tables.

Value

a ggplot2 object

Examples

```
library(ggplot2, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# comparing two analytical methods, in substitute for two conditions
exp_a <- hela_proteins %>%
     expression(knockdown/control) %>%
     export_analysis(knockdown/control, .analysis = "expression")

exp_b <- hela_proteins %>%
     expression(knockdown/control, .method = "limma") %>%
     export_analysis(knockdown/control, .analysis = "expression")

plot_compexp(exp_a, exp_b, log2fc_min = 1, significance_column = "p_value") +
     ggplot2::labs(x = "(log2 FC) Wilcoxon Rank Sum",
                   y = "(log2 FC) Emperical Bayes (limma)",
                   title = "Hela p97 Knockdown ~ Control")
```

plot_counts *Plot the accounting of proteins. peptides, and other counts*

Description

`plot_counts()` is a GGplot2 implementation for plotting counting statistics.

Usage

```
plot_counts(
  data = NULL,
  accounting = NULL,
  show_replicates = TRUE,
  impute_max = 0.5,
  palette = "YlGnBu",
  ...
)
```

Arguments

`data`	tidyproteomics data object
`accounting`	character string
`show_replicates`	boolean to visualize replicates
`impute_max`	a numeric representing the largest allowable imputation percentage
`palette`	a string representing the palette for scale_fill_brewer()
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>% plot_counts()

hela_proteins %>% plot_counts(show_replicates = FALSE, palette = 'Blues')
```

`plot_dynamic_range` *Plot CVs by abundance*

Description

`plot_dynamic_range()` is a GGplot2 implementation for plotting the normalization effects on CVs by abundance, visualized as a 2d density plot. Layered on top is a loess smoothed regression of the CVs by abundance, with the median CV shown in *red* and the dynamic range represented as a box plot on top. The point of this plot is to examine how CVs were minimized through out the abundance profile. Some normalization methods function well at high abundance yet leave retain high CVs at lower abundance.

Usage

```
plot_dynamic_range(data = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
  normalize(.method = c("linear", "loess", "randomforest")) %>%
  plot_dynamic_range()
```

`plot_enrichment` *Bubble plot of enrichment values*

Description

`plot_enrichment()` is a GGplot2 implementation for plotting the enrichment values. This function can take either a tidyproteomics data object or a table with the required headers.

Usage

```
plot_enrichment(
  data = NULL,
  ...,
  .term = NULL,
  enrichment_min = 1,
  enrichment_column = "enrichment",
  significance_max = 0.01,
  significance_column = "p_value",
  term_column = "annotation",
  size_column = "size",
  destination = "plot",
  height = 5,
  width = 8
)
```

Arguments

`data`	a tidyproteomics data object
`...`	two sample comparison
`.term`	a character string indicating the term enrichment analysis should be calculated for
`enrichment_min`	a numeric defining the minimum log2 enrichment to highlight.
`enrichment_column`	a character defining the column name of enrichment values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`term_column`	a character defining the column name for labeling.
`size_column`	a character defining the column name of term size.
`destination`	a character string
`height`	a numeric
`width`	a numeric

Value

a ggplot2 object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(ggplot2, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control, .method = stats::t.test) %>%
  enrichment(knockdown/control, .terms = 'biological_process', .method = "wilcoxon") %>%
   plot_enrichment(knockdown/control, .term = "biological_process") +
   labs(title = "Hela: Term Enrichment", subtitle = "Knockdown ~ Control")
```

---

plot_euler *GGplot2 extension to plot a Euler diagram*

Description

GGplot2 extension to plot a Euler diagram

Usage

```
plot_euler(data, ...)
```

Arguments

`data`	a tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   subset(imputed == 0) %>%
   plot_euler()

hela_proteins %>%
   subset(imputed == 0) %>%
   subset(cellular_component %like% "cytosol") %>%
   plot_euler()
```

plot_heatmap — *Plot a heatmap of quantitative values by sample*

Description

`plot_heatmap()` is a pheatmap implementation for plotting the commonly visualized quantitative heatmap according to sample. Both the samples and the quantitative values are clustered and visualized.

Usage

```
plot_heatmap(data = NULL, tag = NULL, row_names = FALSE, ...)
```

Arguments

`data`	tidyproteomics data object
`tag`	a character string
`row_names`	a boolean
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
  normalize(.method = c("scaled", "median", "linear", "limma", "loess")) %>%
  select_normalization() %>%
  plot_heatmap()
```

plot_normalization — *Plot normalized values*

Description

`plot_normalization()` is a GGplot2 implementation for plotting the normalization effects visualized as a box plot.

Usage

```
plot_normalization(data = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
  normalize(.method = c("scaled", "median", "linear", "limma", "loess")) %>%
  plot_normalization()
```

`plot_pca` *Plot PCA values*

Description

`plot_pca()` is a GGplot2 implementation for plotting two principal components from a PCA analysis, visualized as a scatter.

Usage

```
plot_pca(
  data = NULL,
  variables = c("PC1", "PC2"),
  labels = TRUE,
  label_size = 3,
  ...
)
```

Arguments

`data`	tidyproteomics data object
`variables`	a character vector of the 2 PCs to plot. Acceptable values include (PC1, PC2, PC3 ... PC9). Default c('PC1','PC2').
`labels`	a boolean
`label_size`	a numeric
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins <- hela_proteins %>%
  normalize(.method = c("scaled", "median", "linear", "limma", "loess")) %>%
  select_normalization()

hela_proteins %>% plot_pca()

# a different PC set
hela_proteins %>% plot_pca(variables = c("PC2", "PC3"))

# a PC scree plot
hela_proteins %>% plot_pca("scree")
```

`plot_proportion` *Plot proportional expression values*

Description

`plot_proportion()` is a GGplot2 implementation for plotting the expression differences as fold-change ~ scaled abundance. This allows for the visualization of selected proteins See also `plot_volcano()`. This function can take either a tidyproteomics data object or a table with the required headers.

Usage

```
plot_proportion(
  data = NULL,
  ...,
  log2fc_column = "log2_foldchange",
  log2fc_min = 2,
  significance_column = "adj_p_value",
  significance_max = 0.05,
  proportion_column = "proportional_expression",
  proportion_min = 0.01,
  labels_column = NULL,
  label_significance = TRUE,
  show_pannels = FALSE,
  show_lines = TRUE,
  show_fc_scale = TRUE,
  point_size = NULL,
  color_positive = "dodgerblue",
  color_negative = "firebrick1",
  destination = "plot",
  height = 5,
  width = 8
)
```

Arguments

`data`	a tidyproteomics data object
`...`	two sample comparison
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`proportion_column`	a character defining the column name of the proportional expression values.
`proportion_min`	a numeric defining the minimum proportional expression to highlight.
`labels_column`	a character defining the column name of the column for labeling.
`label_significance`	a boolean for labeling values below the significance threshold.
`show_pannels`	a boolean for showing colored up/down expression panels.
`show_lines`	a boolean for showing threshold lines.
`show_fc_scale`	a boolean for showing the secondary foldchange scale.
`point_size`	a numeric for shanging the point size.
`color_positive`	a character defining the color for positive (up) expression.
`color_negative`	a character defining the color for negative (down) expression.
`destination`	a character string
`height`	a numeric
`width`	a numeric

Value

a ggplot2 object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   plot_proportion(knockdown/control, log2fc_min = 0.5, significance_column = 'p_value')

# generates the same out come
# hela_proteins %>%
#    expression(knockdown/control) %>%
#    export_analysis(knockdown/control, .analysis = 'expression) %>%
#    plot_proportion(log2fc_min = 0.5, significance_column = 'p_value')

# display the gene name instead
```

```
hela_proteins %>%
   expression(knockdown/control) %>%
  plot_proportion(knockdown/control, log2fc_min = 0.5, significance_column = 'p_value', labels_column = "gene_nam
```

plot_protein *Visualize mapped sequence data*

Description

Visualize mapped sequence data

Usage

```
plot_protein(
  mapped_data = NULL,
  protein = NULL,
  row_length = 50,
  samples = NULL,
  modifications = NULL,
  ncol = NULL,
  nrow = NULL,
  color_sequence = "grey60",
 color_modifications = c("red", "blue", "orange", "skyblue", "purple", "yellow"),
  show_modification_precent = TRUE
)
```

Arguments

`mapped_data`	a tidyproteomics data-object, specifically of sequencing origin
`protein`	a character string
`row_length`	a numeric
`samples`	a character string
`modifications`	a character string
`ncol`	a numeric
`nrow`	a numeric
`color_sequence`	a character string
`color_modifications`	a character vector
`show_modification_precent`	a boolean

Value

a list of protein mappings

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

hela_protein_map <- hela_peptides %>%
   protein_map(fasta = path_to_package_data('fasta'))

hela_protein_map %>% plot_protein('P06576')
```

---

plot_quantrank *Plot the variation in normalized values*

Description

plot_quantrank() is a GGplot2 implementation for plotting the variability in normalized values, generating two facets. The left facet is a plot of CVs for each normalization method. The right facet is a plot of the 95%CI in abundance, essentially the conservative dynamic range. The goal is to select a normalization method that minimizes CVs while also retaining the dynamic range.

Usage

```
plot_quantrank(
  data = NULL,
  accounting = NULL,
  type = c("points", "lines"),
  show_error = TRUE,
  show_rank_scale = FALSE,
  limit_rank = NULL,
  display_subset = NULL,
  display_filter = c("none", "log2_foldchange", "p_value", "adj_p_value"),
  display_cutoff = 1,
  palette = "YlGnBu",
  impute_max = 0.5,
  ...
)
```

Arguments

data	tidyproteomics data object
accounting	character string
type	character string
show_error	a boolean
show_rank_scale	a boolean
limit_rank	a numerical vector of 2

`display_subset`	a string vector of identifiers to highlight
`display_filter`	a numeric between 0 and 1
`display_cutoff`	a numeric between 0 and 1
`palette`	a string representing the palette for scale_fill_brewer()
`impute_max`	a numeric representing the largest allowable imputation percentage
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>% plot_quantrank()

hela_proteins %>% plot_quantrank(type = "lines")

hela_proteins %>% plot_quantrank(display_filter = "log2_foldchange", display_cutoff = 1)

hela_proteins %>% plot_quantrank(limit_rank = c(1,50), show_rank_scale = TRUE)
```

`plot_save` *Helper function for saving plots*

Description

`plot_save` helper function

Usage

```
plot_save(
  plot,
  data,
  file_name,
  destination = c("plot", "save", "png", "svg", "tiff", "jpeg"),
  height = 5,
  width = 8,
  ...
)
```

Arguments

`plot`	a ggplot2 object
`data`	a tidyproteomics data object
`file_name`	a character string
`destination`	a character string
`height`	a numeric
`width`	a numeric
`...`	passthrough ggplot2::ggsave arguments

Value

a ggplot2 object

`plot_variation_cv` *Plot the variation in normalized values*

Description

`plot_variation_cv()` is a GGplot2 implementation for plotting the variability in normalized values, generating two facets. The left facet is a plot of CVs for each normalization method. The right facet is a plot of the 95%CI in abundance, essentially the conservative dynamic range. The goal is to select a normalization method that minimizes CVs while also retaining the dynamic range.

Usage

```
plot_variation_cv(data = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
  normalize(.method = c("scaled", "median", "linear", "limma", "loess")) %>%
  plot_variation_cv()
```

`plot_variation_pca` *Plot the PCA variation in normalized values*

Description

`plot_variation_pca()` is a GGplot2 implementation for plotting the variability in normalized values by PCA analysis, generating two facets. The left facet is a plot of CVs for each normalization method. The right facet is a plot of the 95%CI in abundance, essentially the conservative dynamic range. The goal is to select a normalization method that minimizes CVs while also retaining the dynamic range.

Usage

```
plot_variation_pca(data = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
  normalize(.method = c("linear", "loess", "randomforest")) %>%
  plot_variation_pca()
```

`plot_venn` *GGplot2 extension to plot a Venn diagram*

Description

GGplot2 extension to plot a Venn diagram

Usage

```
plot_venn(data, ...)
```

Arguments

`data`	a tidyproteomics data object
`...`	passthrough for ggsave see `plotting`

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   subset(imputed == 0) %>%
   plot_venn()
```

`plot_volcano` *Volcano plot of expression values*

Description

`plot_volcano()` is a GGplot2 implementation for plotting the expression differences as foldchange ~ statistical significance. See also `plot_proportion()`. This function can take either a tidyproteomics data object or a table with the required headers.

Usage

```
plot_volcano(
  data = NULL,
  ...,
  log2fc_min = 1,
  log2fc_column = "log2_foldchange",
  significance_max = 0.05,
  significance_column = "adj_p_value",
  labels_column = "gene_name",
  show_pannels = TRUE,
  show_lines = TRUE,
  show_fc_scale = TRUE,
  show_title = TRUE,
  show_pval_1 = TRUE,
  point_size = NULL,
  color_positive = "dodgerblue",
  color_negative = "firebrick1",
  destination = "plot",
  height = 5,
  width = 8
)
```

Arguments

`data`	a tibble
`...`	two sample comparison
`log2fc_min`	a numeric defining the minimum log2 foldchange to highlight.
`log2fc_column`	a character defining the column name of the log2 foldchange values.
`significance_max`	a numeric defining the maximum statistical significance to highlight.
`significance_column`	a character defining the column name of the statistical significance values.
`labels_column`	a character defining the column name of the column for labeling.
`show_pannels`	a boolean for showing colored up/down expression panels.
`show_lines`	a boolean for showing threshold lines.
`show_fc_scale`	a boolean for showing the secondary foldchange scale.
`show_title`	input FALSE, TRUE for an auto-generated title or any charcter string.
`show_pval_1`	a boolean for showing expressions with pvalue == 1.
`point_size`	a character reference to a numerical value in the expression table
`color_positive`	a character defining the color for positive (up) expression.
`color_negative`	a character defining the color for negative (down) expression.
`destination`	a character string
`height`	a numeric
`width`	a numeric

Value

a ggplot2 object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   plot_volcano(knockdown/control, log2fc_min = 0.5, significance_column = "p_value")

# generates the same out come
# hela_proteins %>%
#     expression(knockdown/control) %>%
#     export_analysis(knockdown/control, .analysis = "expression") %>%
#     plot_volcano(log2fc_min = 0.5, significance_column = "p_value")

# display the gene name instead
hela_proteins %>%
   expression(knockdown/control) %>%
  plot_volcano(knockdown/control, log2fc_min = 0.5, significance_column = "p_value", labels_column = "gene_name")
```

`print.tidyproteomics` *Tidy-Quant data object print definition*

Description

Tidy-Quant data object print definition

Usage

```
## S3 method for class 'tidyproteomics'
print(x, ...)
```

Arguments

`x`	tidyproteomics data object
`...`	unused legacy

Value

print object summary

`println` *Helper function for printing messages*

Description

Helper function for printing messages

Usage

```
println(name = "", message = "", pad_length = 15)
```

Arguments

`name`	string
`message`	string
`pad_length`	string

Value

console print line

protein_map *Align a peptide data to protein sequences for visualization*

Description

Align a peptide data to protein sequences for visualization

Usage

```
protein_map(data = NULL, fasta_path = NULL)
```

Arguments

`data`	a tidyproteomics data-object, specifically of peptide origin
`fasta_path`	a character string representing the path to a fasta file

Value

a list of protein mappings

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

hela_protein_map <- hela_peptides %>%
   protein_map(fasta = path_to_package_data('fasta'))
```

protein_map_munge *Align a peptide data to protein sequences for visualization*

Description

Align a peptide data to protein sequences for visualization

Usage

```
protein_map_munge(
  mapped_data = NULL,
  protein = NULL,
  row_length = 50,
  samples = NULL,
  modifications = NULL
)
```

Arguments

`mapped_data`	a tidyproteomics data-object, specifically of peptide origin
`protein`	a character string
`row_length`	a numeric
`samples`	a character string
`modifications`	a character string

Value

a plot munged list of protein mappings

`read_data` *Read data by format type*

Description

`read_data()` is a helper function that assumes the format type of the data table by checking the ending of path string

Usage

```
read_data(path = NULL, platform = NULL, analyte = c("peptides", "proteins"))
```

Arguments

`path`	a path character string
`platform`	a character string
`analyte`	a character string

Value

tibble

`read_mzTab` *A helper function for importing peptide table data*

Description

A helper function for importing peptide table data

Usage

```
read_mzTab(path = NULL, analyte = c("peptides", "proteins"))
```

Arguments

`path`	a character string
`analyte`	a character string

Value

a tidyproteomics list data-object

`reassign` *reassign the sample info*

Description

`reassign()` enables editing of the sample descriptive in the experimental table. This function will only replace the sample string and update the replicate number.

Usage

```
reassign(data = NULL, ..., .replace = NULL)
```

Arguments

`data`	a tidyproteomics data-object
`...`	a three part expression (eg. x == a)
`.replace`	a character string

Value

a tidyproteomics data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# check the experiment table
hela_proteins %>% summary("experiment")

# make the modification
hela_proteins %>%
   reassign(sample == "control", .replace = "ct") %>%
   reassign(sample == "knockdown", .replace = "kd") %>%
   summary("sample")

# reassign specific file_ids
hela_proteins %>%
   reassign(sample_file == "f1", .replace = "new") %>%
   reassign(sample_file == "f2", .replace = "new") %>%
   summary("sample")
```

`reverselog_transformation`

Reverse the plot axis for log transformation

Description

Reverse the plot axis for log transformation

Usage

```
reverselog_transformation(base = exp(1))
```

Arguments

base	a numeric

Value

a ggplot scale transformation

rf_parallel *parallel compute function for randomforest*

Description

parallel compute function for randomforest

Usage

```
rf_parallel(df)
```

Arguments

`df`	a tibble of raw and centered values

Value

a tibble

rm.mbr *Remove MBR from the dataset across segments*

Description

`rm.mbr()` function is designed to remove match_between_runs between segments. This function will return a smaller tidyproteomics data-object.

Usage

```
rm.mbr(data = NULL, ..., .groups = c("all", "sample"))
```

Arguments

`data`	tidyproteomics data object
`...`	a three part expression (eg. x == a)
`.groups`	a character string

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

hela_proteins %>%
   summary('sample')

hela_proteins %>%
   rm.mbr(.groups = 'sample') %>%
   summary('sample')
```

---

save_local *Store data locally*

Description

`save_local()` will save the tidyproteomics data-object in the local project, based on the given type in the directory ./data/ as either proteins.rds or peptides.rds. This is a *passthrough* function as it returns the original tidyproteomics data-object.

Usage

```
save_local(data = NULL)
```

Arguments

`data`	tidyproteomics data object

Value

tidyproteomics data object

---

save_table *Write table data locally*

Description

`save_table()` will save a summary tibble in the root directory of the local project, based on the extension given in the file name. This is a *passthrough* function as it returns the original tibble.

Usage

```
save_table(table, file_name = NULL)
```

Arguments

`table`	a tibble
`file_name`	a file name with extensions one of (.csv, .tsv, .rds, .xlsx)

Value

a tibble

Examples

```
#\dontrun{
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>%
   expression(knockdown/control) %>%
   export_analysis(knockdown/control, .analysis = "expression") %>%
   save_table("expression_limma_ko_over_wt.csv")
}
```

`select_normalization` *Select a normalization method*

Description

`select_normalization()` selects the best normalization method base on low CVs, low PCA (PC1), and wide Dynamic Range. This is a *passthrough* function as it returns the original tidyproteomics data-object.

Usage

```
select_normalization(data = NULL, normalization = NULL)
```

Arguments

`data`	tidyproteomics data object
`normalization`	a character string

Value

a tidyproteomics data-object

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins <- hela_proteins %>%
  normalize(.method = c("scaled", "median", "linear", "limma", "loess","randomforest")) %>%
   select_normalization()
```

set_vect *set a named vector*

Description

set a named vector

Usage

```
set_vect(config = NULL, category = NULL)
```

Arguments

config	a data.frame of configuration values
category	a character string

Value

a named vector

show_annotations *Display the current annotation data*

Description

Display the current annotation data

Usage

```
show_annotations(data, term = NULL)
```

Arguments

data	tidyproteomics data object
term	a character string

Value

a vector

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

hela_proteins %>% show_annotations()

hela_proteins %>% show_annotations('reactome_pathway')
```

`stats_contamination` *Assess the relative amount of protein contamination*

Description

`stats_contamination()` is an analysis function that can take a regular expression as a means to assign subsets of proteins as contaminant.

Usage

```
stats_contamination(data = NULL, pattern = "CRAP")
```

Arguments

`data`	tidyproteomics data object
`pattern`	character string, regular expression

Value

a tibble

`stats_print` *Helper function for displaying data*

Description

Helper function for displaying data

Usage

```
stats_print(table, title = NULL)
```

Arguments

`table`	a tibble
`title`	a character string

Value

print the table to console

stats_summary *Summarize the protein accounting*

Description

`stats_summary()` is an analysis function that computes the protein summary statistics for a given tidyproteomics data object.

Usage

```
stats_summary(
  data,
  group_by = c("global", "sample", "replicate", "experiment")
)
```

Arguments

`data`	tidyproteomics data object
`group_by`	what to summarize

Value

a tibble

str_normalize *Normalize the column names in a tibble*

Description

Normalize the column names in a tibble

Usage

```
str_normalize(x)
```

Arguments

`x`	a vector

Value

a vector

subset.tidyproteomics *Create a data subset*

Description

subset() is the main function for sub-setting quantitative data from a tidyproteomics data-object based on a regular expression and targeted annotation. This function will return a smaller tidyproteomics data-object.

Note: rm.mbr() is run as default, this is to remove MBR proteins that may no longer have the original "anchor" observation present.

Usage

```
## S3 method for class 'tidyproteomics'
subset(data = NULL, ..., rm.mbr = TRUE, .verbose = TRUE)
```

Arguments

data	tidyproteomics data object
...	a three part expression (eg. x == a)
rm.mbr	a boolean
.verbose	a boolean

Value

a tibble

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# creates a subset of just Ribosomes, based on the string in the annotation
# protein_description
hela_proteins %>%
   subset(description %like% "Ribosome") %>%
   summary()

# creates a subset without Ribosomes
hela_proteins %>%
   subset(!description %like% "Ribosome") %>%
   summary()
```

summary.tidyproteomics

Summarize the data

Description

summary() is an analysis function that computes the protein summary statistics for a given tidyproteomics data object. This is a *passthrough* function as it returns the original tidyproteomics data-object.

Usage

```
## S3 method for class 'tidyproteomics'
summary(object, ...)
```

Arguments

object	tidyproteomics data object
...	passthrough arguments

Value

a tibble on *print*, a tidyproteomics data-object on *save*

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)

# a global summary
hela_proteins %>% summary()

# a summary by sample
hela_proteins %>% summary("sample")

# a summary by sample with imputations removed
hela_proteins %>%
   subset(imputed == 0) %>%
   summary("sample")

# a summary of imputation
hela_proteins %>% summary("imputed")

hela_proteins %>% summary("cellular_component")

hela_proteins %>% summary("biological_process")
```

`svm_parallel` *parallel compute function for randomforest*

Description

parallel compute function for randomforest

Usage

```
svm_parallel(df)
```

Arguments

`df`	a tibble of raw and centered values

Value

a tibble

`table_quantrank` *Helper function to quantitation plots*

Description

`table_quantrank()`

Usage

```
table_quantrank(
  data = NULL,
  accounting = NULL,
  display_filter = c("none", "log2_foldchange", "p_value", "adj_p_value")
)
```

Arguments

`data`	tidyproteomics data object
`accounting`	character string
`display_filter`	a numeric between 0 and 1

Value

a (tidyproteomics data-object | ggplot-object)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tidyproteomics)
hela_proteins %>% plot_quantrank()

hela_proteins %>% plot_quantrank(type = 'lines')

hela_proteins %>% plot_quantrank(type = 'lines', display_filter = 'log2_foldchange', display_cutoff = 1)
```

`theme_palette` *helper function for having nice colors*

Description

helper function for having nice colors

Usage

```
theme_palette(n = 16)
```

Value

character vector of curated html colors

`tidyproteomics` *Tidy-Quant data object print definition*

Description

Tidy-Quant data object print definition

Usage

```
tidyproteomics(obj)
```

Arguments

`obj`	tidyproteomics data object

Value

print object summary

`tidyproteomics_quo` *Helper function to subset a data frame*

Description

Helper function to subset a data frame

Usage

```
tidyproteomics_quo(...)
```

Arguments

`...`	a quo

Value

a list object

`tidyproteomics_quo_name` *Helper function to get a name from the ...*

Description

Helper function to get a name from the ...

Usage

```
tidyproteomics_quo_name(..., sep = "-")
```

Arguments

`...`	a quo

Value

a character string

tidyproteomics_summary *Helper function to summarize the data*

Description

summary() is an analysis function that computes the protein summary statistics for a given tidyproteomics data object. This is a *passthrough* function as it returns the original tidyproteomics data-object.

Usage

```
tidyproteomics_summary(
  data,
  by = c("global"),
  destination = c("print", "save", "return"),
  limit = 25,
  contamination = NULL
)
```

Arguments

data	tidyproteomics data object
by	what to summarize
destination	character string, one of (save, print)
limit	a numeric to limit the number of output groups
contamination	as character string

Value

a tibble on *print*, a tidyproteomics data-object on *save*

transform_factor *helper function for normalizing quantitative data from a tidyproteomics data-object*

Description

helper function for normalizing quantitative data from a tidyproteomics data-object

Usage

```
transform_factor(data, data_factor = NULL, ...)
```

Arguments

`data`	tidyproteomics data object
`data_factor`	tidyproteomics data object
`...`	pass through arguments

Value

a tibble

`transform_log2` *helper function for normalizing a quantitative table*

Description

helper function for normalizing a quantitative table

Usage

```
transform_log2(table, values = "abundance")
```

Arguments

`table`	a tibble
`values`	a character string

Value

a tibble

`transform_median` *helper function for normalizing quantitative data from a tidyproteomics data-object*

Description

helper function for normalizing quantitative data from a tidyproteomics data-object

Usage

```
transform_median(data, group_by = c("identifier"), rename = "log2_med")
```

Arguments

`data`	tidyproteomics data object
`group_by`	character vector
`rename`	character string

Value

a tibble

`write_local` *Helper functio to write data table locally*

Description

`write_local()` will save the data table in the local project,

Usage

```
write_local(table = NULL, file_name = NULL)
```

Arguments

`table`	a tibble
`file_name`	a tibble

Value

tidyproteomics data object

`%like%` *Helper function for subsetting*

Description

Helper function for subsetting

Usage

```
a %like% b
```

Arguments

`a`	a dplyr tibble column reference
`b`	a dplyr tibble column reference

Value

a character string

Index