Package 'TraianProt'

August 11, 2026

Title TraianProt: a user-friendly R package for wide format proteomics data downstream analysis

Version 0.99.17

Description A proteomics data analysis platform that enables the analysis of both label-free and labeled data from Data-Dependent or Data-Independent Acquisition mass spectrometry mode, supporting MaxQuant, MSFragger, DIA-NN, ProteoScape, and Proteome Discoverer output formats. TraianProt provides a comprehensive suite of stepwise downstream analysis modules, which includes data filtering, normalization procedures, and missing value imputation strategies. The platform also incorporates robust statistical frameworks for differential expression testing, with peptide-spectrum match level correction, thereby enhancing the reliability of biological interpretations.

License GPL (>= 3)

biocViews Software, Proteomics, MassSpectrometry, DataImport, Normalization, DifferentialExpression, GO

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/SamueldelaCamaraFuentes/TraianProt

BugReports https://github.com/SamueldelaCamaraFuentes/TraianProt/issues

Imports dplyr, ggplot2, ggrepel, ggExtra, VennDiagram, pheatmap, VIM, grid, wrProteo, wrMisc, gplots, gprofiler2, writexl, readxl, data.table, igraph, stringr, limma, methods, grDevices, graphics, stats, matrixStats, tidyr, tibble, DEqMS, plotly, DOSE, enrichplot, STRINGdb, Rtsne, shiny, shinyWidgets, shinydashboard, DT, rmarkdown, clusterProfiler, SummarizedExperiment

Suggests knitr, BiocStyle, testthat

VignetteBuilder knitr

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

git_branch devel

git_last_commit 8317e3d

git_last_commit_date 2026-07-30

Repository Bioconductor 3.24

Date/Publication 2026-08-10

Author Samuel de la Camara Fuentes [aut, cre] (ORCID: <https://orcid.org/0000-0001-6718-5896>)

Maintainer Samuel de la Camara Fuentes <sdelacam@ucm.es>

Contents

barplot_func *Create Bar Plot from Enrichment Results*

Description

Generates a bar plot from an `enrichResult` object, faceted by cluster.

Usage

```
barplot_func(terms, number, conditions, ...)
```

Arguments

`terms`	A list, where the second element (`terms[[2]]`) is an `enrichResult` object.
`number`	Numeric. The number of top terms *per condition* to display.
`conditions`	Character. A title for the plot.
`...`	Additional arguments passed to `enrichplot::barplot`.

Value

A ggplot object.

Examples

```
# This example requires a valid enrichResult object.
# We will create a minimal synthetic one.
if (requireNamespace("DOSE") && requireNamespace("methods")) {
    dummy_data <- data.frame(
        Cluster = c("Up-regulated", "Down-regulated"),
        Category = "BP", ID = c("GO:0006915", "GO:0007049"),
        Description = c("apoptosis", "cell cycle"),
        p.value = c(0.01, 0.02), adj.P.Val = c(0.01, 0.02),
        query_size = c(50, 40), Count = c(5, 8), term_size = c(100, 120),
        effective_domain_size = 10000, geneID = c("P04637", "P10415"),
        GeneRatio = c("5/50", "8/40"), BgRatio = c("100/10000", "120/10000"),
        Conditions = c("Up-regulated", "Down-regulated")
    )
    dummy_enrich_result <- methods::new("enrichResult", result = dummy_data)
    terms_list <- list(NULL, dummy_enrich_result)
    barplot_func(terms_list, number = 1, conditions = "My Plot")
}
```

boxplot_function *Create Boxplot of Sample Intensities*

Description

Generates a ggplot2 boxplot of LOG2 intensities, grouped by sample and colored by condition.

Usage

```
boxplot_function(df, metadata, selected_conditions)
```

Arguments

df	A data frame containing the LOG2 intensity data.
metadata	A metadata data frame. Must contain 'group' and 'log2_col' columns to map samples to groups.
selected_conditions	A character vector of length 2 specifying the two conditions to compare.

Value

A ggplot object.

Examples

```
# Create synthetic data
log_df <- data.frame(
    LOG2.C1 = rnorm(20, 10),
    LOG2.C2 = rnorm(20, 10.1),
    LOG2.T1 = rnorm(20, 12),
    LOG2.T2 = rnorm(20, 12.1)
)

meta <- data.frame(
    group = c("Control", "Control", "Treatment", "Treatment"),
    log2_col = c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2")
)

# Generate the boxplot
boxplot_function(log_df, meta, selected_conditions = c("Control", "Treatment"))
```

collapse_technical_replicates *Collapse Technical Replicates in Proteomics Data*

Description

This function identifies technical replicates based on a biological replicate metadata column and collapses them by calculating the mean intensity. It updates both the expression data and the associated metadata.

Usage

```
collapse_technical_replicates(df, metadata)
```

Arguments

df	A data.frame or matrix where rows are proteins/features and columns are samples (intensities).
metadata	A data.frame containing at least two columns: log2_col (matching colnames in df) and bioreplicate (grouping ID).

Details

The function assumes that technical replicates share the same sample_name ID in the metadata. It cleans the column names by removing trailing numeric indices (e.g., "Sample, 1" becomes "Sample") using regex. If only one technical replicate exists for a biological group, it simply renames the column.

Value

A list with two elements:

- data: A data.frame with collapsed biological replicates as columns.
- metadata: A data.frame with one row per biological replicate, updated to match the new column names.

Examples

```
# Example metadata
meta <- data.frame(
    log2_col = c("S1, 1", "S1, 2", "S2, 1"),
    sample_name = c("Bio1", "Bio1", "Bio2"),
    intensity_sample_name = c("S1, 1", "S1, 2", "S2, 1"),
    stringsAsFactors = FALSE
)
# Example data
dat <- data.frame(
    `S1, 1` = c(1, 2), `S1, 2` = c(1.2, 2.2), `S2, 1` = c(5, 6),
    check.names = FALSE
)
result <- collapse_technical_replicates(dat, meta)
```

---

corrplot_function *Correlation Matrix*

Description

Calculates a Pearson correlation matrix from a data frame and visualizes it as a tile-based heatmap using ggplot2, overlaying the correlation coefficients.

Usage

```
corrplot_function(
  df,
  metadata,
  display = "circle",
  tl.col = "black",
  addCoef.col = "black"
)
```

Arguments

df	A data frame. The columns for which correlations are desired should be numeric. Non-numeric columns will be ignored by `cor()`.
metadata	A metadata data frame. Must contain 'group' and 'log2_col' columns to map samples to groups.
display	A string specifying the visual representation (e.g., "circle"). **Note:** This parameter is currently not implemented in the function body, which defaults to 'tile'.
tl.col	A string for text label color (i.e., the variable names). **Note:** This parameter is currently not implemented in the function body.
addCoef.col	A string specifying the color for the overlaid correlation coefficients (e.g., `"black"`).

Details

Generate a ggplot2 Correlation Matrix Heatmap

The function first calculates the correlation matrix using `cor()` with `use = "complete.obs"`, which handles missing values by pairwise deletion. It then melts the matrix into a "long" format suitable for `ggplot2`. The plot uses `scale_fill_gradient2` to create a divergent color scale (Red -> White -> Blue) centered at 0. `coord_fixed()` ensures square tiles.

Value

A `ggplot` object representing the correlation heatmap.

Examples

```
df_sample <- data.frame(
    LOG2.C1 = rnorm(20, mean = 10),
    LOG2.C2 = rnorm(20, mean = 10),
    LOG2.T1 = rnorm(20, mean = 12),
    LOG2.T2 = rnorm(20, mean = 12)
)

meta <- data.frame(
    group = c("Control", "Control", "Treatment", "Treatment"),
    log2_col = c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2")
)

# 2. Generate the plot
p <- corrplot_function(df_sample, meta, addCoef.col = "blue")

# 3. Display the plot (Important: in examples we just call the object)
p
```

`Diferential_boxplot` *Differential Protein Boxplot*

Description

Creates a boxplot with jittered points for a single protein, comparing two conditions.

Usage

```
Diferential_boxplot(
  df,
  metadata,
  protein = NULL,
  LOG2.names,
  selected_conditions
)
```

Arguments

`df`	A data frame with expression data. Must have a 'Protein' column and LOG2 intensity columns.
`metadata`	A data frame with sample metadata. Must have 'group' and 'log2_col' columns.
`protein`	A character string specifying the single protein (from the 'Protein' column) to plot.
`LOG2.names`	A character vector of all LOG2 sample column names to use.
`selected_conditions`	A character vector of length 2 specifying the two groups from the 'group' column to compare.

Value

This function is called for its side effect of printing a ggplot. It invisibly returns the ggplot object `p`.

Examples

```
# Create synthetic data
df_box <- data.frame(
    Protein = "Prot1",
    LOG2.C1 = rnorm(1, 8), LOG2.C2 = rnorm(1, 8),
    LOG2.T1 = rnorm(1, 9), LOG2.T2 = rnorm(1, 9)
)
metadata_box <- data.frame(
    log2_col = c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2"),
    group = c("Control", "Control", "Treatment", "Treatment")
)
log_names <- c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2")
selected_cond <- c("Treatment", "Control")

# Generate the plot
if (requireNamespace("DOSE")) {
    Diferential_boxplot(df_box, metadata_box, "Prot1", log_names, selected_cond)
}
```

dotplot_func *Create Dot Plot from Enrichment Results*

Description

Generates a dot plot from a compareClusterResult object, faceted by condition.

Usage

```
dotplot_func(terms, ...)
```

Arguments

terms	A list, where the first element (terms[[1]]) is a compareClusterResult object.
...	Additional arguments passed to enrichplot::dotplot.

Value

A ggplot object.

Examples

```
if (requireNamespace("clusterProfiler", quietly = TRUE)) {
    dummy_data <- data.frame(
        Cluster = "Up-regulated",
        Category = "BP",
        ID = "GO:0006915",
        Description = "apoptosis",
        p.value = 0.01,
        p.adjust = 0.01,
        Count = 5,
        GeneRatio = "5/50",
        BgRatio = "100/10000",
        Conditions = "Up-regulated"
    )

    dummy_cc_result <- methods::new("compareClusterResult",
        compareClusterResult = dummy_data
    )
    dotplot_func(list(dummy_cc_result))
}
```

`filter_valids` *Filter proteins by valid values*

Description

A helper function to filter a protein expression data frame (e.g., from MaxQuant) based on a minimum number of valid (non-NA, non-zero) values across samples.

Usage

```
filter_valids(
  df,
  metadata,
  unique_proteins,
  min_prop = NULL,
  at_least_one = FALSE,
  labeltype = 1
)
```

Arguments

`df`	A data frame containing the raw protein expression data.
`metadata`	A data frame mapping intensity column names to groups.
`unique_proteins`	a list containing unique proteins output from obtain_unique_proteins function .
`min_prop`	A numeric threshold for the minimum number of valid values. For example, 2 means a protein must be valid in at least 2 samples to be kept.
`at_least_one`	Considers whether the condition must be met with both or only one condition.
`labeltype`	Type of mass spectrometry experiment.

Value

A list containing the filtered data frame ("filtered") and the LOG2-transformed data frame ("LOG2").

Examples

```
raw_df <- data.frame(
    Protein = c("P1", "P2", "P3", "P4"),
    LOG2_A1 = c(20, 21, NA, 22),
    LOG2_A2 = c(20, NA, NA, 22),
    LOG2_B1 = c(25, 24, 23, NA),
    LOG2_B2 = c(25, 24, 23, 26)
)
meta_df <- data.frame(
    intensity_sample_name = c("A1", "A2", "B1", "B2"),
    group = c("Control", "Control", "Treatment", "Treatment"),
    log2_col = c("LOG2_A1", "LOG2_A2", "LOG2_B1", "LOG2_B2")
)
unique_list <- list(
    data.frame(Protein = character()),
    data.frame(Protein = character())
```

```
)

filtered_data <- filter_valids(
    df = raw_df,
    metadata = meta_df,
    unique_proteins = unique_list,
    min_prop = 0.5,
    at_least_one = FALSE,
    labeltype = 1
)

head(filtered_data)
```

`gostplot_func` *Manhattan Plot for Functional Enrichment*

Description

This function is a helper wrapper that extracts the raw gprofiler2 results (the third element) from the output of the Goterms_finder function to generate a Manhattan plot of enriched terms.

Usage

```
gostplot_func(terms, ...)
```

Arguments

`terms`	A list of length 3 generated by Goterms_finder function.
`...`	Additional arguments passed to `gprofiler2::gostplot`

Value

A ggplot or plotly object representing the Manhattan plot of functional enrichment.

Examples

```
gostplot_func(res)
```

`Goterms_finder` *Find GO Terms (gprofiler2)*

Description

Performs Gene Ontology (GO) and pathway enrichment analysis for up- and down-regulated proteins using gprofiler2.

Usage

```
Goterms_finder(
  df,
  raw,
  target,
  numeric_ns,
  mthreshold,
  filter_na,
  organismo,
  custombg,
  platform,
  ...
)
```

Arguments

df	A data frame from statistical analysis (e.g., limma), must have 'expression', 'p.value', 'Protein', and/or 'Protein_description' columns.
raw	A data frame of the "raw" data, used to build the background gene list. Must have 'Protein' or 'Protein_description'.
target	Character. The target namespace for gconvert (e.g., "ENSG").
numeric_ns	Character. Namespace for numeric IDs.
mthreshold	Numeric. g:GOSt multi-query significance threshold.
filter_na	Logical. Whether gconvert should filter out NA results.
organismo	Character. The organism name for gprofiler2 (e.g., "hsapiens").
custombg	Logical. Whether to use a custom background (from raw).
platform	Numeric. An indicator for the data platform (1, 2, 3, or 4).
...	Additional arguments passed to gprofiler2::gost.

Value

A list named go_structures containing: 1. A compareClusterResult object (for plotting). 2. An enrichResult object (for plotting). 3. The raw gost result object (for gostplot).

Examples

```
limma_df <- data.frame(
    Protein = c("P04637", "P00533", "P01116"),
    Protein_description = c("TP53", "EGFR", "KRAS"),
    expression = c("Up-regulated", "Up-regulated", "Up-regulated"),
    p.value = c(0.001, 0.002, 0.003)
)

# raw_df is needed for the function structure, even if custombg=FALSE
raw_df <- data.frame(
    Protein = c("P04637", "P00533", "P01116", "P62258"),
    Protein_description = c("TP53", "EGFR", "KRAS", "ACTB")
)

# 2. Run the function with checks
# This example requires an internet connection
```

```
if (requireNamespace("gprofiler2", quietly = TRUE)) {
    try(
        {
            go_res <- Goterms_finder(
                df = limma_df,
                raw = raw_df,
                target = "ENSG",
                numeric_ns = "",
                mthreshold = 0.05,
                filter_na = TRUE,
                organismo = "hsapiens",
                custombg = FALSE,
                platform = 1
            )

            # 3. Safe printing (checks if result exists before printing)
            # We check if go_res exists, is a list, and has the expected slot
            if (!is.null(go_res) &&
                is.list(go_res) &&
                length(go_res) >= 1 &&
                inherits(go_res[[1]], "compareClusterResult")) {
                print(head(as.data.frame(go_res[[1]])))
            }
        },
        silent = TRUE
    )
}
```

`histogram` *Plot a Histogram*

Description

A simple wrapper to plot a histogram for a specific column in a data frame.

Usage

```
histogram(df, colname, color, title)
```

Arguments

`df`	A data frame.
`colname`	The name of the (numeric) column to plot.
`color`	The color for the histogram.
`title`	The main title for the plot.

Value

A histogram plot.

Examples

```
my_data <- data.frame(p.value = rnorm(100))
histogram(my_data, "p.value", "lightblue", "P-Value Distribution")
```

`identify_proteins` *Identify Proteins*

Description

Identifies unique and common proteins between two data frames.

Usage

```
identify_proteins(raw, metadata, platform, selected_conditions)
```

Arguments

`raw`	A data frame containing the raw protein expression data.
`metadata`	A data frame mapping intensity column names to groups.
`platform`	An indicator (e.g., 1) for the data platform (e.g., MaxQuant).
`selected_conditions`	Conditions compared from you experiment.

Value

A list containing three data frames: `unique_df1` (proteins unique to df1), `unique_df2` (proteins unique to df2), and `common_df` (proteins common to both).

Examples

```
metadata_example <- data.frame(
    intensity_sample_name = c("Log_C1", "Log_C2", "Log_T1", "Log_T2"),
    group = c("Control", "Control", "Treatment", "Treatment")
)

raw_example <- data.frame(
    Protein = c("Prot_A", "Prot_B", "Prot_C", "Prot_D"),
    Log_C1 = c(25.1, 24.2, 0, 22.0),
    Log_C2 = c(25.0, 24.5, 0, 22.2),
    Log_T1 = c(0, 26.1, 23.5, 21.9),
    Log_T2 = c(0, 26.0, 23.8, 22.1)
)
plot_result <- identify_proteins(
    raw = raw_example,
    metadata = metadata_example,
    platform = 1,
    selected_conditions = c("Control", "Treatment")
)
print(plot_result)
```

`igraph_analysis` *Perform igraph Analysis on STRING Subnetworks*

Description

Calculates graph theory measures for up- and down-regulated protein networks.

Usage

```
igraph_analysis(interactions, taxonid, score)
```

Arguments

`interactions`	A list of data frames (like that returned by `interactions_up`) containing mapped STRING IDs. `interactions[[1]]` = up-regulated, `interactions[[2]]` = down-regulated.
`taxonid`	Numeric. The NCBI taxonomy ID for the species (e.g., 9606 for human).
`score`	Numeric. The minimum required interaction score (0-1000).

Value

A list of two data frames, `Upregulated` and `Downregulated`, containing graph measures for each network.

Examples

```
if (requireNamespace("STRINGdb") && requireNamespace("igraph")) {
    # Create synthetic data
    up_mapped <- data.frame(
        Protein = c("TP53", "EGFR"),
        STRING_id = c("9606.ENSP00000269305", "9606.ENSP00000275493")
    )
    down_mapped <- data.frame(
        Protein = c("CALCA", "ACTB"),
        STRING_id = c("9606.ENSP00000302220", "9606.ENSP00000482455")
    )
    interactions_list <- list(up_mapped, down_mapped)

    # Run analysis
    analysis_results <- igraph_analysis(interactions_list, 9606, 400)
    print(analysis_results$Upregulated)
}
```

impute_data *Impute Missing Values using Down-Shifted Normal Distribution*

Description

Replaces non-finite (NA, Inf) values in specified LOG2 columns with random values drawn from a normal distribution. This distribution is shifted to a lower mean and narrower standard deviation relative to the observed valid data.

Usage

```
impute_data(df, LOG2.names, width = 0.3, downshift = 1.8)
```

Arguments

df	A data frame. This frame **must** contain a logical column named KEEP which specifies the rows to be used for calculating the mean and standard deviation of the valid data.
LOG2.names	A character vector of column names (from df) to impute.
width	Numeric. The factor by which to shrink the standard deviation of the valid data (e.g., 0.3).
downshift	Numeric. The factor (in standard deviations) by which to shift the mean of the valid data (e.g., 1.8).

Details

This method is a common "down-shift" or "Perseus-style" imputation for label-free proteomics, simulating the low-abundance signals of proteins near the detection limit. It assumes missingness is not at random (MNAR).

Value

The data frame df with imputed LOG2 columns and new 'impute.*' logical columns indicating which values were imputed.

Examples

```
# Create synthetic LOG2 data with non-finite values (Inf)
log_df <- data.frame(
    Protein = c("ProtA", "ProtB", "ProtC", "ProtD"),
    LOG2.S1 = c(10, 12, Inf, 8),
    LOG2.S2 = c(11, 13, Inf, Inf),
    # Add a KEEP column (e.g., from a filter)
    # Here, we keep all proteins for calculation
    KEEP = c(TRUE, TRUE, TRUE, TRUE)
)

# Specify columns to impute
log_cols <- c("LOG2.S1", "LOG2.S2")

# Run imputation
set.seed(123) # for reproducible example
```

```
imputed_df <- impute_data(log_df, log_cols)

print("Original Data:")
print(log_df)
print("Imputed Data:")
print(imputed_df)
```

`impute_KNN_data` *Impute missing data using k-NN*

Description

Uses K-Nearest Neighbors (k-NN) to impute missing (NA) values in a log-expression matrix.

Usage

```
impute_KNN_data(df, LOG2.names, ...)
```

Arguments

`df`	A numeric matrix with NA values.
`LOG2.names`	A character vector of column names (from `df`) to impute.
`...`	Additional arguments passed to the `plot` function.

Value

A numeric matrix with NA values imputed.

Examples

```
log_matrix_na <- matrix(
    c(
        10.0, 10.2, NA, 10.5, 10.1, 9.9, # Sample 1 (with NAs)
        12.0, NA, 11.9, 12.1, 11.8, NA
    ), # Sample 2 (with NAs)
    nrow = 6, ncol = 2,
    dimnames = list(
        c("P1", "P2", "P3", "P4", "P5", "P6"),
        c("Sample1", "Sample2")
    )
)

LOG2.names <- c("Sample1", "Sample2")

imputed_matrix <- impute_KNN_data(as.data.frame(log_matrix_na), LOG2.names)

print(imputed_matrix)
```

`interactions_down` *Plot STRING Interactions for Down-regulated Proteins*

Description

Maps down-regulated proteins to the STRING database and plots the network.

Usage

```
interactions_down(df, taxonid, score)
```

Arguments

`df`	A data frame from statistical analysis (e.g., limma), must have 'expression' and 'Protein' columns.
`taxonid`	Numeric. The NCBI taxonomy ID for the species (e.g., 9606 for human).
`score`	Numeric. The minimum required interaction score (0-1000).

Value

This function is called for its side effect of generating a plot. It does not return an object.

Examples

```
if (requireNamespace("STRINGdb")) {
    # Create synthetic data
    limma_df <- data.frame(
        Protein = c("P10415", "P62258"), # CALCA, ACTB
        expression = c("Down-regulated", "Down-regulated")
    )
    # Run function
    interactions_down(limma_df, taxonid = 9606, score = 400)
}
```

`interactions_up` *Get STRING Interactions for Up-regulated Proteins*

Description

Maps up-regulated proteins to the STRING database, plots the network, and returns a list of mapped proteins.

Usage

```
interactions_up(df, taxonid, score)
```

Arguments

df	A data frame from statistical analysis (e.g., limma), must have 'expression' and 'Protein' columns.
taxonid	Numeric. The NCBI taxonomy ID for the species (e.g., 9606 for human).
score	Numeric. The minimum required interaction score (0-1000).

Value

A list containing two data frames: 1. up_mapped: Mapped up-regulated proteins. 2. down_mapped: Mapped down-regulated proteins.

Examples

```
if (requireNamespace("STRINGdb")) {
    # Create synthetic data
    limma_df <- data.frame(
        Protein = c("P04637", "P00533"), # TP53, EGFR
        expression = c("Up-regulated", "Up-regulated")
    )
    # Run function
    interactions_list <- interactions_up(limma_df, taxonid = 9606, score = 400)
}
```

---

median_centering *Perform median centering normalization*

Description

Normalizes a numeric matrix of log-expression values by subtracting the median of each column.

Usage

```
median_centering(df, LOG2.names)
```

Arguments

df	A numeric matrix where rows are proteins and columns are samples.
LOG2.names	A vector containing the samples with log2 intensity values.

Value

A numeric matrix of the same dimensions, now median-centered.

Examples

```
# Create a small, synthetic log-expression matrix
log_df <- data.frame(
    Sample1 = c(10.0, 10.2, 9.8),
    Sample2 = c(12.0, 12.1, 11.9)
)
row.names(log_df) <- c("ProtA", "ProtB", "ProtC")

LOG2.names <- c("Sample1", "Sample2")

normalized_df <- median_centering(log_df, LOG2.names)

print(normalized_df)
```

`my_heatmap` *Base R Heatmap*

Description

Creates a heatmap using the base R `heatmap` function.

Usage

```
my_heatmap(data, cond.names, title)
```

Arguments

`data`	A data frame containing expression data and a 'Protein' column.
`cond.names`	A character vector of the column names (samples) to include in the heatmap.
`title`	A character string for the heatmap's title.

Value

Returns (invisibly) a list with components from the `heatmap` function, such as `rowInd` and `colInd`.

Examples

```
# Create synthetic data
heat_data <- data.frame(
    Protein = paste0("Prot", 1:5),
    Sample1 = rnorm(5, 10),
    Sample2 = rnorm(5, 12)
)
sample_names <- c("Sample1", "Sample2")

# Generate the plot
my_heatmap(heat_data, sample_names, "My Heatmap")
```

my_heatmap_differential *Differential Protein Heatmap (ggplot)*

Description

Creates a ggplot2 heatmap of differentially expressed proteins.

Usage

```
my_heatmap_differential(limma, data, cond.names, title)
```

Arguments

limma	A data frame from statistical analysis, must have 'Protein' and 'expression' columns (e.g., 'Up-regulated', 'Down-regulated').
data	A data frame containing expression data, must have 'Protein' and sample columns (e.g., LOG2 intensity values).
cond.names	A character vector of the column names (samples) to include in the heatmap.
title	A character string for the heatmap's title.

Value

A ggplot object.

Examples

```
# Create synthetic data
limma_results <- data.frame(
    Protein = c("Prot1", "Prot2", "Prot3"),
    expression = c("Up-regulated", "Not Significant", "Down-regulated"),
    logFC = c(2.5, 0.5, -2.1)
)
data_expr <- data.frame(
    Protein = c("Prot1", "Prot2", "Prot3"),
    Sample1 = c(12, 10, 8),
    Sample2 = c(13, 10, 7)
)
sample_names <- c("Sample1", "Sample2")

# Generate the plot
my_heatmap_differential(limma_results, data_expr, sample_names, "DE Heatmap")
```

`normalization_func` *Apply Normalization to LOG2 Columns*

Description

A wrapper function to apply various normalization methods from the `wrMisc` package to the specified LOG2 intensity columns.

Usage

```
normalization_func(df, LOG2.names, method)
```

Arguments

`df`	A data frame containing the LOG2 expression data.
`LOG2.names`	A character vector of column names (from `df`) to which the normalization should be applied.
`method`	A character string specifying the normalization method to be passed to `wrMisc::normalizeThis`. Examples include "median", "mean", "trimMean", "quant", etc.

Value

The original data frame (`df`) with the specified `LOG2.names` columns replaced by their normalized versions.

Examples

```
# Create synthetic LOG2 data
log_df <- data.frame(
    Protein = c("ProtA", "ProtB"),
    LOG2.Sample1 = c(10, 12),
    LOG2.Sample2 = c(11, 13)
)

# Specify which columns to normalize
log_cols <- c("LOG2.Sample1", "LOG2.Sample2")

# Run median normalization
normalized_df <- normalization_func(log_df, log_cols, "median")

print("Original Data:")
print(log_df)
print("Normalized Data:")
print(normalized_df)
```

obtain_LOG.names *Obtain LOG2 Column Names*

Description

A helper function to find all column names containing "LOG2".

Usage

```
obtain_LOG.names(df)
```

Arguments

df	A data.frame to search.

Value

A character vector of column names.

Examples

```
# Create a small example data frame
my_df <- data.frame(
    Protein = c("P1", "P2"),
    Intensity.A = c(100, 200),
    LOG2.A = c(6.6, 7.6),
    Intensity.B = c(110, 210),
    LOG2.B = c(6.7, 7.7)
)

# This example is runnable!
log_names <- obtain_LOG.names(my_df)
print(log_names)
```

obtain_unique_proteins *Obtain Unique Proteins Between Conditions*

Description

Identifies proteins that are uniquely present in one of two conditions, based on the presence of finite (e.g., LOG2 intensity) vs. non-finite (e.g., Inf, NA) values.

Usage

```
obtain_unique_proteins(df, metadata, selected_conditions)
```

Arguments

df	A data frame containing expression data, with LOG2 intensity columns and a 'Protein' column.
metadata	A data frame with sample metadata. Must have 'group' and 'log2_col' columns.
selected_conditions	A character vector of length 2 specifying the two groups from the 'group' column to compare.

Details

This function is designed for label-free data where a protein's absence is marked by a non-finite value (like Inf from a log-transformation of 0). It identifies proteins present in at least half of the replicates of one condition while being completely absent (non-finite) in the other.

Value

A list containing two data frames: 1. `cond1_unicas`: A data frame of proteins unique to the first group. 2. `cond2_unicas`: A data frame of proteins unique to the second group.

Examples

```
# Create synthetic data with infinite values (representing missing)
df_prot <- data.frame(
    Protein = paste0("Prot", 1:4),
    LOG2.C1 = c(10, 12, Inf, 9),
    LOG2.C2 = c(11, 13, Inf, 10),
    LOG2.T1 = c(Inf, Inf, 8, 11),
    LOG2.T2 = c(Inf, Inf, 7, 12)
)
# Prot1/2 are present in Control, absent in Treatment
# Prot3 is present in Treatment, absent in Control
# Prot4 is present in both

metadata <- data.frame(
    log2_col = c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2"),
    group = c("Control", "Control", "Treatment", "Treatment")
)
# Note: T is [1] (second_group), C is [2] (first_group)
selected_cond <- c("Treatment", "Control")

unique_list <- obtain_unique_proteins(df_prot, metadata, selected_cond)

# cond1_unicas (Control) should have Prot1, Prot2
print("Unique to Control:")
print(unique_list[[1]]$Protein)

# cond2_unicas (Treatment) should have Prot3
print("Unique to Treatment:")
print(unique_list[[2]]$Protein)
```

pca *Custom PCA Plot (Base R)*

Description

Performs PCA and generates a 2D scatter plot of the first two principal components using base R plotting.

Usage

```
pca(x, metadata, selected_conditions, pc_x = 1, pc_y = 2)
```

Arguments

x	A data frame or matrix with expression data. Columns are samples (e.g., LOG2 intensity values). Rows are features (e.g. proteins).
metadata	A data frame with sample metadata. Must have 'group' and 'log2_col' columns.
selected_conditions	A character vector of length 2 specifying the two groups from the 'group' column to compare.
pc_x	Principal component for X axis.
pc_y	Principal component for Y axis.

Value

This function is called for its side effect of generating a base R plot. It does not return an object.

Examples

```
# Create synthetic data
x_data <- data.frame(
    LOG2.C1 = rnorm(10, 8), LOG2.C2 = rnorm(10, 8),
    LOG2.T1 = rnorm(10, 9), LOG2.T2 = rnorm(10, 9),
    row.names = paste0("Prot", 1:10)
)
metadata <- data.frame(
    log2_col = c("LOG2.C1", "LOG2.C2", "LOG2.T1", "LOG2.T2"),
    group = c("Control", "Control", "Treatment", "Treatment")
)
selected_cond <- c("Treatment", "Control") # Matches user's logic [2] then [1]

# Generate the plot
pca(x_data, metadata, selected_cond)
```

plotCV2 *Plot Coefficient of Variation vs. Mean*

Description

Generates a plot of the squared Coefficient of Variation (CV) versus the mean abundance (A). This is often used to check imputation quality.

Usage

```
plotCV2(y, trend = TRUE, main = "Imputation check", ...)
```

Arguments

y	A numeric data frame or matrix where rows are features (proteins) and columns are samples (LOG2 intensities).
trend	Logical. If TRUE, a limma::loessFit trendline is added to the plot.
main	Character. The main title for the plot.
...	Additional arguments passed to the plot function.

Value

A data frame containing the mean (A) and CV (CV^2) values that were plotted.

Examples

```
# Create synthetic LOG2 intensity data
log_data <- matrix(rnorm(100, mean = 10, sd = 1), nrow = 20, ncol = 5)
log_data[sample(1:100, 10)] <- NA # Add some NAs

# Create the CV plot
cv_data <- plotCV2(log_data, main = "CV vs. Mean Plot")
print(head(cv_data))
```

postimputation_state *Plot Post-Imputation Density*

Description

Plots the density of intensity values before and after imputation to visualize the effect of the imputation.

Usage

```
postimputation_state(data_filtered, imputation, LOG2.names)
```

Arguments

data_filtered	The data frame *before* imputation, containing non-finite values (NA, Inf).
imputation	An indicator (1 or 2) specifying the imputation method. 1 = impute_KNN_data, 2 = impute_data.
LOG2.names	A character vector of the LOG2 column names to use.

Value

A ggplot object showing the two density plots.

Examples

```
# This function depends on `impute_KNN_data` and `impute_data`
# We must create those functions and the data they need.

# 1. Create synthetic data
data_filt <- data.frame(
    Protein = c("A", "B", "C", "D"),
    LOG2.S1 = c(10, 12, NA, 8),
    LOG2.S2 = c(11, 13, 9, NA),
    KEEP = c(TRUE, TRUE, TRUE, TRUE) # For impute_data
)

# 2. Create minimal versions of the imputation functions for the example
impute_KNN_data <- function(df, lnames, k) {
    df[lnames][is.na(df[lnames])] <- mean(unlist(df[lnames]), na.rm = TRUE)
    return(df)
}
impute_data <- function(df, lnames) {
    df[lnames][is.na(df[lnames])] <- rnorm(sum(is.na(df[lnames])), mean = 7, sd = 0.3)
    return(df)
}

# 3. Run the plot function
postimputation_state(data_filt, imputation = 2, LOG2.names = c("LOG2.S1", "LOG2.S2"))
```

---

preimputation_state *Plot Missingness Pattern*

Description

Generates a missing value aggregation plot from the VIM package to visualize patterns of missing data.

Usage

```
preimputation_state(df, cond.names)
```

Arguments

df	A data frame containing the data to be plotted.
cond.names	A character vector of column names from df to be included in the missingness plot.

Value

A plot showing the aggregation of missing values.

Examples

```
# Create synthetic data with NAs
log_df <- data.frame(
    LOG2.C1 = c(10, 10.1, NA, 9.9),
    LOG2.C2 = c(11, NA, NA, 11.2),
    LOG2.T1 = c(12, 12.1, 12.2, 12.3)
)

# Plot the missingness pattern
preimputation_state(log_df, c("LOG2.C1", "LOG2.C2", "LOG2.T1"))
```

---

qqplot_function *Create a Q-Q Plot*

Description

Generates a Q-Q plot to compare the distributions of two columns in a data frame.

Usage

```
qqplot_function(df, colname1, colname2, color)
```

Arguments

df	A data frame.
colname1	Character. The name of the column for the x-axis.
colname2	Character. The name of the column for the y-axis.
color	The color for the points.

Value

A ggplot object.

Examples

```
my_data <- data.frame(
    Theoretical = rnorm(100),
    Sample = rnorm(100, mean = 0.5)
)
qqplot_function(my_data, "Theoretical", "Sample", "blue")
```

quick_filtering *Quick Data Filtering*

Description

Filters raw proteomics data (e.g., from MaxQuant) to remove contaminants, reverse hits, and site-only identifications. It also parses protein identifiers from Fasta headers.

Usage

```
quick_filtering(
  raw,
  platform,
  organism,
  metadata,
  selected_conditions,
  directory_path = NULL
)
```

Arguments

raw	A data frame of raw proteomics data (e.g., from MaxQuant).
platform	An indicator (e.g., 1) for the data platform (e.g., MaxQuant).
organism	An indicator (e.g., 1 for Human, 2 for Mouse, etc.).
metadata	A data frame mapping intensity column names to sample names.
selected_conditions	(Not currently used in function) A vector of selected conditions.
directory_path	(Not currently used in function) A path to a directory.

Value

A data frame filtered to remove contaminants/reverse hits, with new "Protein" and "Protein_description" columns.

Examples

```
raw_data <- data.frame(
    Potential.contaminant = c("", "+", "", ""),
    Reverse = c("", "", "+", ""),
    Only.identified.by.site = c("", "", "", "+"),
    Fasta.headers = c(
        ">sp|P04637|P53_HUMAN Cellular tumor antigen p53 OS=Homo sapiens",
        ">sp|P00000|CONTAM_Protein Common contaminant",
        ">sp|P12345|REV_Protein Reverse Sequence",
        ">sp|P62258|1433E_HUMAN 14-3-3 protein epsilon"
    ),
    # Intensity columns
    Intensity.S1 = c(10000, 500, 200, 100),
    Intensity.S2 = c(12000, 600, 150, 90),
    stringsAsFactors = FALSE
)
```

```
metadata <- data.frame(
    intensity_sample_name = c("Intensity.S1", "Intensity.S2"),
    sample_name = c("Sample_A", "Sample_B")
)

filtered_df <- quick_filtering(
    raw = raw_data,
    platform = 1,
    organism = 2,
    metadata = metadata,
    selected_conditions = NULL,
    directory_path = NULL
)
```

`runTraianProt` *Launch the TraianProt Shiny Application*

Description

This function builds and returns the Shiny app object for TraianProt.

Usage

```
runTraianProt()
```

Value

A `shiny.appobj` object representing the TraianProt app.

Examples

```
app <- runTraianProt()

if (interactive()) {
    shiny::runApp(app)
}
```

`scatterplot_function` *Create a Scatterplot with Marginal Densities*

Description

Uses `ggplot2` and `ggExtra` to create a scatterplot comparing two columns, with marginal density plots.

Usage

```
scatterplot_function(df, colname1, colname2)
```

Arguments

df	A data frame.
colname1	Character. The name of the column for the x-axis.
colname2	Character. The name of the column for the y-axis.

Value

A ggExtra plot object.

Examples

```
my_data <- data.frame(
    Sample1 = rnorm(100),
    Sample2 = rnorm(100)
)
scatterplot_function(my_data, "Sample1", "Sample2")
```

---

statistical_analysis *Limma function for statistical analysis*

Description

A comprehensive function to perform differential expression (DE) analysis between two conditions. It supports t-tests, Wilcoxon tests, and limma, with options for paired data and DEqMS spectral count weighting.

Usage

```
statistical_analysis(
  df,
  test,
  paired = FALSE,
  metadata,
  logfc,
  sig,
  adjval,
  statval,
  unique_proteins,
  way,
  psms,
  platform,
  selected_conditions,
  diann_dir = NULL
)
```

Arguments

df	The primary data frame containing expression data (rows=proteins, cols=samples). Must also contain 'Protein' and 'Protein_description' columns.
test	Numeric. The statistical test to use: 1 for t.test, 2 for limma, 3 for wilcox.test.
paired	Logical. TRUE if the experimental design is paired, FALSE otherwise.
metadata	A data frame with sample metadata. Must contain group (condition) and log2_col (sample names matching df colnames).
logfc	Numeric. The log-fold change (LFC) threshold for 'Up-regulated' and 'Down-regulated' status.
sig	Numeric. The significance threshold (e.g., 0.05) to be used with statval.
adjval	String. The p-value adjustment method for p.adjust (e.g., "BH").
statval	Numeric. Which p-value to use for significance: 1 for raw p-value, 2 for adjusted p-value.
unique_proteins	A list with two elements containing proteins unique to the control ([1]) and treatment ([2]) groups. Used only if way = 2.
way	Numeric. 1 to analyze only shared proteins, 2 to include unique proteins from unique_proteins.
psms	Logical. If test = 2 (limma), TRUE to use DEqMS for spectral count weighting.
platform	Numeric. Used if psms = TRUE. Specifies data source for finding PSM columns: 1=MaxQuant, 2=MSFragger, 4=Proteome Discoverer.
selected_conditions	A character vector of length 2. selected_conditions[2] is treated as the control group, and selected_conditions[1] as the treatment group.
diann_dir	String. Directory path for DIA-NN. **Note:** This parameter is currently not implemented in the function body.

Details

Perform Differential Expression Statistical Analysis

This function acts as a wrapper for various statistical tests. The logFC is calculated as mean(treatment) - mean(control).

When way = 2, unique proteins are added to the results with imputed logFC and p-values (e.g., p-value = min(p.value)/100) to ensure they appear in visualizations.

The function uses several non-standard variable assignment methods (e.g., assign, ls, get) to dynamically create sample groups.

Value

A data frame (results.eb) with DE results, including logFC, p.value, adj.P.Val, expression, Protein, and Protein_description. Additional columns may be present for limma/DEqMS (e.g., sca.P.Value).

Examples

```
# 1. Create a mock data frame (df)
set.seed(123)
df_data <- data.frame(
    Protein = paste0("Prot", 1:5),
    Protein_description = paste("Description for", paste0("Prot", 1:5)),
    Cond1_R1 = rnorm(5, 10, 1),
    Cond1_R2 = rnorm(5, 10, 1),
    Cond2_R1 = rnorm(5, 12, 1), # Cond2 is higher
    Cond2_R2 = rnorm(5, 12, 1)
)
rownames(df_data) <- df_data$Protein

# 2. Create mock metadata
meta <- data.frame(
    log2_col = c("Cond1_R1", "Cond1_R2", "Cond2_R1", "Cond2_R2"),
    group = c("Condition1", "Condition1", "Condition2", "Condition2")
)

# 3. Define unique proteins (for way = 2)
uniques_list <- list(
    data.frame(Protein = "Prot_Control", Protein_description = "Only in Control"),
    data.frame(Protein = "Prot_Treat", Protein_description = "Only in Treatment")
)

# Run analysis (t-test, unpaired, no PSMs, include uniques)
results <- statistical_analysis(
    df = df_data,
    test = 1, # t-test
    paired = FALSE,
    metadata = meta,
    logfc = 1,
    sig = 0.05,
    adjval = "BH",
    statval = 2, # Use adjusted p-value
    unique_proteins = uniques_list,
    way = 2, # Include uniques
    psms = FALSE,
    platform = 1,
    selected_conditions = c("Condition2", "Condition1") # Treat 2 vs 1
)

print(results)
```

`traianprot_power_curve` *Create power curve*

Description

Calculates the theoretical power curve based on the observed variability in the experimental data and a desired Fold Change.

Usage

```
traianprot_power_curve(
  df,
  log2_cols,
  foldchange,
  replicatespower,
  alpha_level_choice,
  alpha_level
)
```

Arguments

df	A data.frame or matrix containing the expression data.
log2_cols	Character vector or numeric indices indicating which columns of df contain the abundance values (must be numeric).
foldchange	Numeric. The expected Fold Change to calculate the power (e.g., 2 for doubling, 0.5 for halving).
replicatespower	Numeric. Number of replicates per group.
alpha_level_choice	Numeric. Choice to consider either p-value or adjusted p-value.
alpha_level	Numeric. Global significance level before correction (defaults to 0.05).

Value

A ggplot object containing the power curve.

Examples

```
# Simulated data
data <- data.frame(
    ID = letters[1:10],
    R1 = rnorm(10), R2 = rnorm(10), R3 = rnorm(10)
)
plot <- traianprot_power_curve(
    data, c("R1", "R2", "R3"),
    foldchange = 2,
    replicatespower = 5,
    alpha_level_choice = 1,
    alpha_level = 0.05
)
print(plot)
```

---

traian_to_SE *Export TraianProt results to SummarizedExperiment*

Description

Compiles the raw, normalized, and statistical results from the TraianProt pipeline into a Bioconductor-standard SummarizedExperiment object.

Usage

```
traian_to_SE(df.F, total_dataset, metadata, df_limma)
```

Arguments

df.F	dataframe from preprocessing methdos.
total_dataset	whole dataframe including abscence and presence proteins
metadata	metadata that defines the experiment
df_limma	dataframe containing statistical results

Value

a Bioconductor-standard SummarizedExperiment object, containing the raw, normalized and statistical resutls.

Examples

```
df_raw <- data.frame(
    Protein = paste0("Protein_", 1:5),
    WT_1 = runif(5, 100, 1000), WT_2 = runif(5, 100, 1000),
    Treat_1 = runif(5, 100, 1000), Treat_2 = runif(5, 100, 1000)
)

df_norm <- df_raw
df_norm[, 2:5] <- log2(df_norm[, 2:5]) # Simulamos la normalización

meta <- data.frame(
    Condition = c("WT", "WT", "Treatment", "Treatment"),
    log2_col = c("WT_1", "WT_2", "Treat_1", "Treat_2")
)

stats <- data.frame(
    Protein = paste0("Protein_", 1:5),
    logFC = c(1.2, -0.5, 2.1, 0.1, -1.8),
    p_value = c(0.01, 0.4, 0.005, 0.8, 0.02)
)

se_object <- traian_to_SE(df_raw, df_norm, meta, stats)
se_object
```

---

tsne *t-SNE Plot*

Description

Generates a t-SNE plot for sample clustering using ggplot2.

Usage

```
tsne(x, metadata, perplexity_num, selected_conditions)
```

Arguments

x	A data frame or matrix with expression data. Columns are samples (e.g., LOG2 intensity values). Rows are features (e.g. proteins).
metadata	A data frame with sample metadata. Must have 'group' and 'log2_col' columns.
perplexity_num	Numeric. The perplexity value for the t-SNE algorithm.
selected_conditions	A character vector of length 2 specifying the two groups from the 'group' column to compare.

Value

A ggplot object.

Examples

```
df <- data.frame(
    # Group A (Low values)
    Sample1 = c(10.1, 10.5, 12.0, 10.0),
    Sample2 = c(10.2, 10.6, 12.1, 10.1),
    Sample3 = c(10.3, 10.7, 12.2, 10.2),
    Sample4 = c(10.4, 10.8, 12.3, 10.3),
    Sample5 = c(10.5, 10.9, 12.4, 10.4),
    # Group B (High values)
    Sample6 = c(20.1, 21.5, 22.0, 20.0),
    Sample7 = c(20.2, 21.6, 22.1, 20.1),
    Sample8 = c(20.3, 21.7, 22.2, 20.2),
    Sample9 = c(20.4, 21.8, 22.3, 20.3),
    Sample10 = c(20.5, 21.9, 22.4, 20.4)
)
row.names(df) <- c("P1", "P2", "P3", "P4")

metadata <- data.frame(
    log2_col = paste0("Sample", 1:10),
    group = c(rep("A", 5), rep("B", 5))
)
selected_conditions <- c("A", "B")

if (requireNamespace("Rtsne") && requireNamespace("ggplot2")) {
    tsne_plot <- tsne(
        x = df,
        metadata = metadata,
        perplexity_num = 2,
        selected_conditions = selected_conditions
    )

    print(tsne_plot)
}
```

---

unique_peptides_filter

Filter Proteins by Unique Peptide Counts

Description

Filters a protein data frame based on a minimum number of unique peptides found in a minimum fraction of replicates per condition.

Usage

```
unique_peptides_filter(df, metadata, number, min_fraction)
```

Arguments

df	A data frame with protein data. Must have rownames (e.g., Protein IDs) and columns corresponding to unique peptide counts.
metadata	A metadata data frame. Must contain 'group' and 'unique_peptides_col' columns to map samples to groups.
number	Numeric. The minimum number of unique peptides (e.g., 1 or 2) required to count as 'present' in a single sample.
min_fraction	Numeric. The minimum fraction (0 to 1) of replicates in *at least one* group that must meet the 'number' threshold. (e.g., 0.5 means at least 50% of replicates).

Details

This function is useful for filtering data (e.g., from MaxQuant) to remove proteins with low confidence. It keeps a protein if, for **at least one** experimental group (condition), the unique peptide count is `>= number` in at least `min_fraction` of the samples in that group.

Value

A data frame (`df_filtered`) containing only the rows (proteins) that pass the filter.

Examples

```
# Create synthetic data
peptide_df <- data.frame(
    Pep.C1 = c(2, 0, 1, 5),
    Pep.C2 = c(3, 1, 0, 6),
    Pep.T1 = c(0, 2, 1, 0),
    Pep.T2 = c(0, 3, 0, 0)
)
rownames(peptide_df) <- c("ProtA", "ProtB", "ProtC", "ProtD")

meta <- data.frame(
    unique_peptides_col = c("Pep.C1", "Pep.C2", "Pep.T1", "Pep.T2"),
    group = c("Control", "Control", "Treatment", "Treatment")
)

# Filter: require min 1 peptide in at least 75% of reps (i.e., 2/2)
# ProtA: Passes in Control (2/2 have >= 1). Kept.
# ProtB: Passes in Treatment (2/2 have >= 1). Kept.
# ProtC: Fails in Control (1/2), Fails in Treatment (1/2). Dropped.
# ProtD: Passes in Control (2/2 have >= 1). Kept.

filtered_df <- unique_peptides_filter(
    df = peptide_df,
    metadata = meta,
```

```
    number = 1,
    min_fraction = 0.75
)

print("Filtered proteins (should be A, B, D):")
print(rownames(filtered_df))
```

`venn_diagram` *Create a Venn Diagram*

Description

Generates a Venn diagram plot object from a list of unique proteins and a data frame of common proteins.

Usage

```
venn_diagram(df, unique_proteins, color1, color2)
```

Arguments

`df`	A data frame of common proteins (must have a "Protein" column).
`unique_proteins`	A list containing two data frames: 1. Proteins unique to the first set. 2. Proteins unique to the second set.
`color1`	Color for the first set.
`color2`	Color for the second set.

Value

A gList object (a plot) representing the Venn diagram.

Examples

```
common_proteins <- data.frame(
    Protein = c("ProteinA", "ProteinB", "ProteinC")
)

unique_protein_list <- list(
    Control = data.frame(Protein = c("ProteinD", "ProteinE")),
    Treatment = data.frame(Protein = c("ProteinF", "ProteinG", "ProteinH", "ProteinI"))
)

my_venn_plot <- venn_diagram(
    df = common_proteins,
    unique_proteins = unique_protein_list,
    color1 = "blue",
    color2 = "red"
)

grid::grid.newpage()
grid::grid.draw(my_venn_plot)
```

volcano_plot *This function generates an interactive volcano plot using the 'plotly' package based on differential expression results.*

Description

This function generates an interactive volcano plot using the 'plotly' package based on differential expression results.

Usage

```
volcano_plot(limma, title, label, statval, psms)
```

Arguments

limma	A data frame containing differential expression results. This data frame must include columns: • logFC: Log-fold change values. • Protein: Protein identifiers (used for hover-text). • expression: A character vector (e.g., "Up-regulated", "Down-regulated", "Unchanged") used for coloring. • p.value or sca.P.Value: Nominal p-values. • adj.P.Val or sca.adj.pval: Adjusted p-values.
title	A character string for the plot title.
label	A numeric value specifying the size of the markers (points) in the plot. Passed to size = I(label).
statval	A numeric value (1 or 2) specifying which p-value to plot on the y-axis. • 1: Plots the nominal p-value (e.g., p.value or sca.P.Value). • 2: Plots the adjusted p-value (e.g., adj.P.Val or sca.adj.pval).
psms	A logical value (TRUE or FALSE). • TRUE: Uses PSM-aware p-values from DEqMS (i.e., sca.P.Value or sca.adj.pval). • FALSE: Uses standard p-values (i.e., p.value or adj.P.Val).

Details

The function selects the appropriate p-value column based on the statval and psms arguments. The y-axis label is automatically set to "-log10 p-value" or "-log10 q-value" (for adjusted p-values). Points are colored based on the expression column.

Value

A plotly object, which renders as an interactive volcano plot.

Examples

```
# Create a mock data frame for demonstration
limma_results <- data.frame(
    logFC = rnorm(100, 0, 2),
    Protein = paste0("PROT", 1:100),
    p.value = runif(100, 0, 1),
    adj.P.Val = runif(100, 0, 1),
    sca.P.Value = runif(100, 0, 1),
    sca.adj.pval = runif(100, 0, 1),
    expression = sample(c("Up-regulated", "Down-regulated", "Unchanged"),
        100,
        replace = TRUE
    )
)

# Generate an interactive plot using adjusted p-values without PSM weighting
volcano_plot(
    limma = limma_results,
    title = "Interactive Volcano Plot",
    label = 5,
    statval = 2,
    psms = FALSE
)
```

`volcano_plot_tiff` *Create a Static Volcano Plot for Publication*

Description

This function generates a static volcano plot using the 'ggplot2' package, suitable for saving as a high-resolution image (e.g., TIFF) for publication.

Usage

```
volcano_plot_tiff(limma, title, label, statval, psms)
```

Arguments

`limma`	A data frame containing differential expression results. This data frame must include columns: • `logFC` or `log2FC`: Log-fold change values. • `Protein`: Protein identifiers. • `expression`: A character vector (e.g., "Up-regulated", "Down-regulated", "Unchanged") used for coloring. • `p.value` or `sca.P.Value`: Nominal p-values. • `adj.P.Val` or `sca.adj.pval`: Adjusted p-values.
`title`	A character string for the plot title.
`label`	A numeric value specifying the size of the points. **Note:** In the current function, this parameter is only used when `statval = 1` and `psms = TRUE`. In all other cases, the size is hard-coded to 2.

`statval` A numeric value (1 or 2) specifying which p-value to plot on the y-axis.

- `1`: Plots the nominal p-value (e.g., `p.value` or `sca.P.Value`).
- `2`: Plots the adjusted p-value (e.g., `adj.P.Val` or `sca.adj.pval`).

`psms` A logical value (`TRUE` or `FALSE`).

- `TRUE`: Uses PSM-aware p-values from DEqMS (i.e., `sca.P.Value` or `sca.adj.pval`).
- `FALSE`: Uses standard p-values (i.e., `p.value` or `adj.P.Val`).

Details

The function uses `ggplot2::theme_light()` and customizes axis text and line sizes. The y-axis label is automatically set to "-log10 p-value" or "-log10 q-value" (for adjusted p-values).

Value

A `ggplot` object, which renders as a static volcano plot.

Examples

```
mock_fc <- rnorm(100, 0, 2)

limma_results <- data.frame(
    logFC = mock_fc,
    log2FC = mock_fc, # Use the vector we just created
    Protein = paste0("PROT", 1:100),
    p.value = runif(100, 0, 1),
    adj.P.Val = runif(100, 0, 1),
    sca.P.Value = runif(100, 0, 1),
    sca.adj.pval = runif(100, 0, 1),
    expression = sample(c("Up-regulated", "Down-regulated", "Unchanged"),
        100,
        replace = TRUE
    )
)

# Generate a static plot
static_plot <- volcano_plot_tiff(
    limma = limma_results,
    title = "Static Volcano Plot",
    label = 3,
    statval = 2,
    psms = TRUE
)

print(static_plot)
```

Index