

Package: NADIA (via r-universe)

October 6, 2026

Type Package

Title Missing Value-Aware Differential Abundance Analysis of DIA
Proteomics Data

Version 0.99.1

Description Provides an end-to-end workflow for differential abundance analysis of protein quantification data, with particular attention to missing values in data-independent acquisition (DIA) proteomics. Quantification reports exported from Spectronaut, DIA-NN and Proteome Discoverer for TMT and label-free experiments (DDA/DIA) are converted into a common data structure, normalised, optionally batch-corrected, imputed and analysed using limma or limpa. The package includes individual and two-stage imputation strategies that treat values assumed to be missing at random and missing not at random separately. Data-driven functions evaluate and rank normalisation and imputation workflows using simulated missingness or spike-in datasets with known ground truth, allowing method selection to be justified for each dataset. Results can be explored through interactive Highcharts visualisations, static ggplot2 and ComplexHeatmap figures, and sortable tables.

License GPL (>= 3)

Encoding UTF-8

URL <https://github.com/sciordia/NADIA>,
<https://sciordia.github.io/NADIA/>

BugReports <https://github.com/sciordia/NADIA/issues>

biocViews Software, Proteomics, MassSpectrometry,
DifferentialExpression, Normalization, QualityControl,
BatchEffect

Depends R (>= 4.5.0)

Imports dplyr, ggplot2, grDevices, grid, highcharter, htmltools,
htmlwidgets, jsonlite, magrittr, methods, paletteer,

RColorBrewer, reactable, readr, rlang, S4Vectors, stats,
stringr, SummarizedExperiment, tibble, tidyr, tidyselect,
tools, utils

Suggests arrow, BERT, Biobase, BiocStyle, circlize, cluster,
ComplexHeatmap, crosstalk, DBI, duckdb, e1071, ggsci, impute,
imputeLCMD, knitr, limma, limpa, lme4, MASS, Mfuzz, mice,
missForest, norm, pcaMethods, pROC, rmarkdown, rrcovNA, scales,
testthat (>= 3.0.0), tidyHeatmap, vegan, viridis, vsn, withr

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/Needs/website pkgdown

Config/roxygen2/version 8.1.0

Config/pak/sysreqs

cmake libgmp-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-09-06 11:57:57 UTC

RemoteUrl <https://github.com/BiocStaging/NADIA>

RemoteRef HEAD

RemoteSha c4a896085c0e2b7966e6ddf55af3f07aa3a4c114

Contents

batch_correct_proteomics	5
benchmark_aucBars_gg	7
benchmark_confusion_gg	8
benchmark_confusion_overall_gg	9
benchmark_heatmap_gg	10
benchmark_metricsBars_gg	11
benchmark_roc_gg	12
benchmark_signifBars_gg	13
benchmark_volcano_hc	14
benchmark_volcano_hc_list	15
benchmarking_multiple	16
benchmarking_proteomics	19
bm_compute_extended_ranking	21
bm_compute_ranking	22
bm_compute_ranking_by_comparison	23
bm_plot_confusion_stacked	24
bm_plot_extended_rankingBars	25
bm_plot_extended_ranking_heatmap	26
bm_plot_metrics_comparison	27
bm_plot_metrics_heatmap	28
bm_plot_rankingBars	29

bm_plot_ranking_bars_by_comp	29
bm_plot_ranking_heatmap	30
bm_plot_ranking_heatmap_by_comp	31
bm_plot_roc	32
boxplot_highchart_list	33
build_clustering_matrix	34
build_long_output	35
build_pca_scores	36
cluster_centroids_highchart	37
cluster_profile_highchart	38
cluster_profile_highchart_list	40
compute_benchmark_metrics	41
compute_confidence_ellipse	42
compute_dispersion_metrics	43
compute_hulls	44
compute_opdea_metrics	44
configure_cluster_palette	46
create_expression_set	46
de_analysis_proteomics	47
deps_with_clusters	48
detect_condition_columns	50
evaluate_cluster_range	51
extract_se_data	51
filter_significant_features	52
get_annotation_palette	53
get_heatmap_palette	53
im_compute_metrics	54
im_plot_acc_oi	55
im_plot_metrics	56
im_plot_nrmse	56
im_plot_pss	57
im_plot_ranking	58
im_plot_sor	58
im_prepare_se	59
import_benchmark_metrics	60
import_classified_results	61
import_confusion_results	62
import_imp_matrices	63
import_norm_matrices	64
import_opdea_results	66
imputation_metrics	67
impute_proteomics	68
merge_significance_info	71
nadia_add_pattern_profiler	71
nadia_connect	72
nadia_deps_with_clusters	73
nadia_dia	75
nadia_export_parquet	76

nadia_pattern_profiler	77
nadia_preprocessing	78
nadia_result	79
nadia_skill_path	80
nadia_tables	81
nm_compute_metrics	81
nm_plot_boxplot	83
nm_plot_correlation	84
nm_plot_density	85
nm_plot_final_ranking	85
nm_plot_mds	86
nm_plot_mds1_ranking	87
nm_plot_metrics	88
nm_plot_pc1_ranking	89
nm_plot_pca	90
nm_plot_pcv	91
nm_plot_pev	92
nm_plot_pmad	93
nm_plot_qq	94
nm_plot_scatter	95
nm_prepare_se	96
nm_rank_cor	97
nm_rank_final	98
nm_rank_mds1	99
nm_rank_pc1	100
nm_rank_pcv	101
nm_rank_pev	102
nm_rank_pmad	103
nm_run_normalizations	104
normalization_metrics	105
normalize_proteomics	107
pattern_profiler_analysis	108
pca_covariates_plot	110
pca_highchart	112
pca_highchart_list	114
prepare_heatmap_data	115
preprocess_diann	116
preprocess_lfq	118
preprocess_spectronaut	120
preprocess_tmt	121
print.diann_data	123
print.lfq_data	124
print.nadia_db	125
print.proteomics_heatmap	125
print.proteomics_result	126
print.spectronaut_data	127
print.tmt_data	127
process_proteomics	128

protein_list_reactable	132
protein_list_widget	133
proteomics_heatmap	134
proteomics_heatmap_list	138
pvca_analysis	142
pvca_compute	144
pvca_plot	145
quant_list_widget	146
read_nadia	148
read_pattern_profiler_data	149
results_list_reactable	150
results_list_widget	151
run_mfuzz_clustering	152
select_optimal_clusters	153
standardize_eset	154
summarize_pattern_profiler	154
summary_list_widget	155
volcano_highchart_list	156
wrap_heatmap_with_title	157
write_nadia	158
Index	160

batch_correct_proteomics

Batch Correction with BERT

Description

Applies BERT batch effect correction to a normalized assay in a SummarizedExperiment. BERT uses hierarchical tree decomposition to handle missing values, making it suitable for pre-imputation batch correction on protein-level proteomics data.

Usage

```
batch_correct_proteomics(
  se,
  assay_name,
  batch_column = "Batch",
  corrected_assay_name = "BERT",
  algorithm = "ComBat",
  ComBat_mode = 1,
  covariates = NULL,
  qualitycontrol = FALSE,
  verbose = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment with a normalized assay.
<code>assay_name</code>	Character. Name of the input assay to correct (e.g., "cycloess").
<code>batch_column</code>	Character. Column in <code>colData(se)</code> containing batch assignments (default "Batch").
<code>corrected_assay_name</code>	Character. Name for the new corrected assay added to the SE (default "BERT").
<code>algorithm</code>	Character: "ComBat" (default), "limma", or "ref".
<code>ComBat_mode</code>	Integer 1-4 controlling ComBat behavior: 1 = parametric + mean+variance, 2 = parametric + mean-only, 3 = non-parametric + mean+variance, 4 = non-parametric + mean-only.
<code>covariates</code>	Character vector of column names from <code>colData(se)</code> to protect during batch correction (default NULL). These are mapped to BERT's <code>Cov_1</code> , <code>Cov_2</code> , ... format internally. BERT uses those columns directly as the design matrix and encodes nothing itself, so categorical columns are converted to indicator variables here; numeric columns are passed through unchanged and act as continuous covariates. IMPORTANT: for ComBat/limma, include here the biological variable of interest (e.g. the condition) in order to PRESERVE it; otherwise ComBat removes all batch-associated variance and can erase biological signal when condition and batch are confounded.
<code>qualitycontrol</code>	Logical. Compute ASW (Average Silhouette Width) quality metrics for raw vs corrected data (default FALSE).
<code>verbose</code>	Logical (default TRUE).

Value

SummarizedExperiment with the new corrected assay added. If BERT returns fewer features than the input, the SE is subsetting to match and a warning is issued.

Examples

```
data(nadia_dia)

# nadia_dia has no batch information, so the example builds a plausible one:
# two digestion batches crossed with the three conditions.
cov <- data.frame(Column = nadia_dia$metadata$Coding,
                  Batch = rep(c("b1", "b2"),
                              length.out = nrow(nadia_dia$metadata)),
                  stringsAsFactors = FALSE)
res <- process_proteomics(nadia_dia, covariate_df = cov, verbose = FALSE)

# BERT removes the batch effect and adds a "BERT" assay, leaving the
# original one untouched
se <- batch_correct_proteomics(res$se_proc, assay_name = "Impseqrob_min",
                              batch_column = "Batch", verbose = FALSE)
SummarizedExperiment::assayNames(se)

# On a real experiment you would normally pass the biological variable of
# interest in `covariates`, so that ComBat preserves it instead of removing
```

```
# it along with the batch effect. `covariates = "Condition"` works on this
# example: the assay is already imputed, so each batch-by-condition cell has
# its two values. Before imputation, with only two samples per cell, BERT
# needs two non-missing values per cell and any protein with a gap there
# stops being adjustable.
```

benchmark_aucBars_gg *AUC Bar Chart (ggplot2)*

Description

Horizontal bar chart of AUC per comparison, colored by value, with a reference line at 0.5.

Usage

```
benchmark_aucBars_gg(
  metrics_table,
  title = "AUC by Comparison",
  bar_width = 0.7
)
```

Arguments

metrics_table	Data frame from compute_benchmark_metrics()
title	Plot title
bar_width	Bar width (default: 0.7)

Value

ggplot2 object

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
if (requireNamespace("pROC", quietly = TRUE)) {
  metrics <- compute_benchmark_metrics(de, ev)
  benchmark_aucBars_gg(metrics)
}
```

benchmark_confusion_gg

Confusion Matrix Heatmap (ggplot2)

Description

Heatmap of Comparisons x (TP%, FP%, FN%, TN%) with white-to-blue gradient.

Usage

```
benchmark_confusion_gg(
  confusion_df,
  title = "Confusion Matrix by Comparison x Species",
  text_size = 3.5,
  axis_text_size = 10
)
```

Arguments

confusion_df	Data frame from .confusion_by_species()
title	Plot title
text_size	Size of cell text labels
axis_text_size	Size of axis text

Value

ggplot2 object

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
bench <- benchmarking_proteomics(de, ev, verbose = FALSE)
benchmark_confusion_gg(bench$confusion_by_species)
```

benchmark_confusion_overall_gg

Overall Confusion Matrix Heatmap (ggplot2)

Description

Heatmap of Comparisons x (TP%, FP%, FN%, TN%) aggregated across all species, with white-to-blue gradient fill by percentage.

Usage

```
benchmark_confusion_overall_gg(
  confusion_overall_df,
  title = "Confusion Matrix by Comparison (Overall)",
  text_size = 4,
  axis_text_size = 11
)
```

Arguments

confusion_overall_df	Data frame from .confusion_overall()
title	Plot title
text_size	Size of cell text labels
axis_text_size	Size of axis text

Value

ggplot2 object

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
bench <- benchmarking_proteomics(de, ev, verbose = FALSE)
benchmark_confusion_overall_gg(bench$confusion_overall)
```

 benchmark_heatmap_gg *Performance Heatmap (ggplot2)*

Description

Heatmap of Comparisons x Metrics (AUC, Sensitivity, Specificity, Precision, F1, Accuracy). Gradient red -> yellow -> green.

Usage

```
benchmark_heatmap_gg(
  metrics_table,
  metrics_to_show = c("AUC", "Sensitivity", "Specificity", "Precision", "F1", "Accuracy"),
  title = "Benchmark Performance Heatmap",
  text_size = 4,
  axis_text_size = 11
)
```

Arguments

metrics_table	Data frame from compute_benchmark_metrics()
metrics_to_show	Character vector of metrics to display
title	Plot title
text_size	Size of cell text labels
axis_text_size	Size of axis text

Value

ggplot2 object

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
metrics <- compute_benchmark_metrics(de, ev)
benchmark_heatmap_gg(metrics, metrics_to_show = c("AUC", "F1", "MCC"))
```

benchmark_metricsBars_gg*Grouped Metrics Bar Chart (ggplot2)*

Description

Grouped bar chart: Comparisons x multiple metrics side by side.

Usage

```
benchmark_metricsBars_gg(  
  metrics_table,  
  metrics_to_show = c("Sensitivity", "Specificity", "Precision", "F1", "Accuracy"),  
  title = "Benchmark Metrics by Comparison",  
  bar_width = 0.7  
)
```

Arguments

metrics_table	Data frame from compute_benchmark_metrics()
metrics_to_show	Character vector of metrics to display
title	Plot title
bar_width	Bar width (default: 0.7)

Value

ggplot2 object

Examples

```
data(nadia_dia)  
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results  
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",  
                                   package = "NADIA"))  
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]  
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),  
                 Species = c("ECOLI", "YEAST"),  
                 expected_logFC = c(1, -0.58, 2, -3.3))  
metrics <- compute_benchmark_metrics(de, ev)  
benchmark_metricsBars_gg(metrics, metrics_to_show = c("Sensitivity", "F1"))
```

benchmark_roc_gg

*ROC Curves by Comparison (ggplot2)***Description**

Generates ROC curves for each comparison using `pROC::ggroc()`. Requires the pROC package.

Usage

```
benchmark_roc_gg(
  classified_df,
  p_col = "adj.P.Val",
  comparisons = NULL,
  title = "ROC Curves by Comparison",
  zoom = FALSE,
  palette = "Set1"
)
```

Arguments

<code>classified_df</code>	Classified data frame (output of <code>.classify_all_comparisons</code>)
<code>p_col</code>	P-value column name used as predictor score
<code>comparisons</code>	Comparisons to include (NULL = all)
<code>title</code>	Plot title
<code>zoom</code>	If TRUE, zoom into the low-FPR region (x = 0 to 0.1)
<code>palette</code>	RColorBrewer palette name (default: "Set1")

Value

ggplot2 object or NULL if pROC is not available

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
if (requireNamespace("pROC", quietly = TRUE)) {
  bench <- benchmarking_proteomics(de, ev, verbose = FALSE)
  benchmark_roc_gg(bench$classified_df, comparisons = "B-A")
}
```

benchmark_signif_bars_gg

Significant Proteins Stacked Bars (ggplot2)

Description

Stacked bar chart of significant proteins by species, faceted by direction (UP/DOWN).

Usage

```
benchmark_signif_bars_gg(
  classified_df,
  ev = NULL,
  species_colors = NULL,
  title = "Significant Proteins by Species and Direction"
)
```

Arguments

classified_df	Classified data frame
ev	Ignored, kept for backwards compatibility with earlier calls
species_colors	Named vector of colors per species (optional)
title	Plot title

Details

Counts every protein the pipeline called significant, whatever the direction. A spike-in found significant with the sign inverted is an FN, so it carries predicted = 0, but it is still a significant protein and belongs in the facet its observed logFC points to – which is precisely the facet that reveals the problem.

Value

ggplot2 object

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
bench <- benchmarking_proteomics(de, ev, verbose = FALSE)
benchmark_signif_bars_gg(bench$classified_df, ev)
```

benchmark_volcano_hc	<i>Benchmark Volcano Plot (Highcharter)</i>
----------------------	---

Description

Volcano plot colored by Species with plotLines at expected logFC values, dispersion stats in subtitle.

Usage

```
benchmark_volcano_hc(  
  de_res_comp,  
  ev_comp,  
  disp_comp = NULL,  
  alpha = 0.05,  
  lfc_thr = 0,  
  p_col = "adj.P.Val",  
  species_colors = NULL,  
  point_size = 3,  
  title = NULL,  
  height = NULL  
)
```

Arguments

de_res_comp	DE results for one comparison
ev_comp	Expected values for one comparison
disp_comp	Dispersion metrics for one comparison (optional)
alpha	Significance threshold
lfc_thr	Log fold-change threshold
p_col	P-value column
species_colors	Named vector of colors per species
point_size	Marker radius (default: 3)
title	Custom title (optional)
height	Chart height in pixels (optional)

Value

Highchart object

Examples

```

data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
disp <- compute_dispersion_metrics(de, ev)
benchmark_volcano_hc(de[de$Comparison == "B-A", ],
                     ev[ev$Comparison == "B-A", ],
                     disp_comp = disp[disp$Comparison == "B-A", ])

```

benchmark_volcano_hc_list

List of Benchmark Volcano Plots (Highcharter)

Description

Generates one volcano plot per comparison.

Usage

```

benchmark_volcano_hc_list(
  de_res,
  ev,
  disp = NULL,
  alpha = 0.05,
  lfc_thr = 0,
  p_col = "adj.P.Val",
  comparisons = NULL,
  assay = NULL,
  species_colors = NULL,
  point_size = 3,
  height = NULL,
  species_df = NULL
)

```

Arguments

de_res	Data frame with DE results
ev	Data frame with expected values
disp	Dispersion metrics data frame (optional)
alpha	Significance threshold (default: 0.05)
lfc_thr	Log fold-change threshold (default: 0)

p_col	P-value column (default: "adj.P.Val")
comparisons	Comparisons to include (NULL = all)
assay	Assay to filter (NULL = all)
species_colors	Named vector of colors per species (optional)
point_size	Marker radius (default: 3)
height	Chart height in pixels (optional)
species_df	Data frame with Protein.IDs and Species columns (optional). If NULL, de_res must already contain a Species column.

Value

Named list of highchart objects

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
volcanoes <- benchmark_volcano_hc_list(de, ev)
names(volcanoes)
volcanoes[["B-A"]]
```

benchmarking_multiple *Multiple-method OpDEA Benchmarking*

Description

Orchestrator that imports, ranks, visualizes, and exports results from multiple normalization/imputation method combinations.

Usage

```
benchmarking_multiple(
  opdea_combined = NULL,
  confusion_combined = NULL,
  classified_combined = NULL,
  bench_metrics_combined = NULL,
  results_dir = NULL,
  pattern = "benchmark_opdea_metrics\\|.tsv$",
  method_names = NULL,
  recursive = TRUE,
```

```

strip_prefix = "benchmark_",
metrics = .BM_METRICS,
extended_metrics = .BM_EXTENDED_METRICS,
p_col = "adj.P.Val",
plots = "all",
verbose = TRUE,
output_dir = NULL,
export_plots = TRUE,
export_tables = TRUE,
plot_width = 12,
plot_height = 8,
plot_dpi = 150
)

```

Arguments

<code>opdea_combined</code>	Optional pre-built data.frame with columns: Assay, Comparison, nMCC, G_mean, pAUC_001, pAUC_005, pAUC_010. If NULL, results are imported from <code>results_dir</code> .
<code>confusion_combined</code>	Optional pre-built data.frame with columns: Assay, Comparison, TP, FP, TN, FN. If NULL and <code>results_dir</code> is provided, imported from <code>benchmark_confusion_overall.tsv</code> files.
<code>classified_combined</code>	Optional pre-built data.frame with columns: Assay, Comparison, truth, and a p-value column. If NULL and <code>results_dir</code> is provided, imported from <code>benchmark_classified.tsv</code> files. Used for multi-method ROC curves.
<code>bench_metrics_combined</code>	Optional pre-built data.frame from <code>import_benchmark_metrics()</code> with columns: Assay, Comparison, Sensitivity, Specificity, Precision, NPV, F1, Accuracy, MCC, Performance. If NULL and <code>results_dir</code> is provided, imported from <code>benchmark_metrics.tsv</code> files. Used for extended 11-metric ranking.
<code>results_dir</code>	Parent directory containing benchmark subfolders (each with <code>benchmark_opdea_metrics.tsv</code>). Used only if <code>opdea_combined</code> is NULL.
<code>pattern</code>	Regex for opdea metrics filename (default: <code>"benchmark_opdea_metrics\\.tsv\$"</code>)
<code>method_names</code>	Optional Assay names for file import
<code>recursive</code>	Search subdirectories? (default: TRUE)
<code>strip_prefix</code>	Prefix removed from each Assay name (default <code>"benchmark_"</code>), so folder-derived names in <code>results_dir</code> mode match the in-memory ones (e.g. <code>"benchmark_Rlr_none_BERT"</code> -> <code>"Rlr_none_BERT"</code>). Set to NULL or "" to keep the full folder name. Harmless when the prefix is absent.
<code>metrics</code>	Character vector of metrics for OpDEA ranking (default: all 5 OpDEA metrics)
<code>extended_metrics</code>	Character vector of metrics for extended ranking (default: all 11 classification + OpDEA metrics)
<code>p_col</code>	P-value column name for ROC curves (default: <code>"adj.P.Val"</code>)

plots	Which plots to generate: "all" or character vector of names. Valid names: "ranking_heatmap_mean", "ranking_heatmap_median", "rankingBars_mean", "rankingBars_median", "metrics_heatmap_mean", "metrics_heatmap_median", "metrics_comparison", "extended_rankingBars", "extended_ranking_heatmap"
verbose	Print progress messages (default: TRUE)
output_dir	Directory for exporting results (NULL = no export)
export_plots	Export plots as PNG (default: TRUE)
export_tables	Export tables as TSV (default: TRUE)
plot_width	Plot width in inches (default: 12)
plot_height	Plot height in inches (default: 8)
plot_dpi	Plot resolution (default: 150)

Value

Named list with:

opdea_combined Combined input data.frame
mean_aggregated Mean of metrics per Assay
median_aggregated Median of metrics per Assay
mean_ranking Ranking table (mean-based)
median_ranking Ranking table (median-based)
gg_ranking_heatmap_mean Heatmap of ranks (mean)
gg_ranking_heatmap_median Heatmap of ranks (median)
gg_rankingBars_mean Bar chart of final rank (mean)
gg_rankingBars_median Bar chart of final rank (median)
gg_metrics_heatmap_mean Heatmap of metric values (mean)
gg_metrics_heatmap_median Heatmap of metric values (median)
gg_metrics_comparison Boxplot distributions
ranking_by_comparison Named list of ranking data.frames per comparison
ranking_by_comparison_combined data.frame with all per-comparison rankings
gg_ranking_heatmap_by_comp Named list of heatmaps per comparison
gg_rankingBars_by_comp Named list of bar charts per comparison
confusion_combined Combined confusion data.frame (if available)
gg_confusion_stacked Faceted confusion stacked bars (if available)
gg_confusion_stacked_by_comp Named list of confusion plots per comparison
classified_combined Combined classified data.frame (if available)
gg_roc_by_comp Named list of ROC plots per comparison (if available)
gg_roc_zoom_by_comp Named list of ROC zoom plots per comparison (if available)
extended_combined Merged classification + OpDEA data (if bench_metrics available)
extended_mean_aggregated Mean-aggregated extended metrics per Assay

extended_ranking Extended ranking table (11 metrics, mean-based)
gg_extended_ranking_bars Bar chart of extended ranking
gg_extended_ranking_heatmap Heatmap of extended ranking
parameters List of parameters used

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
root <- file.path(tempdir(), "nadia_bench")
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                          verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  benchmarking_proteomics(de, ev, output_dir = file.path(root, nm),
                          verbose = FALSE)
}
result <- benchmarking_multiple(results_dir = root, verbose = FALSE)
result$mean_ranking
result$gg_ranking_bars_mean
```

benchmarking_proteomics

Benchmark Proteomics Differential Expression Results

Description

Evaluates the reliability of DE results by comparing them against known ground truth (multi-species spike-in experiments). Computes classification metrics, dispersion statistics, and generates visualizations.

Usage

```
benchmarking_proteomics(
  de_res,
  expected_values,
  alpha = 0.05,
  lfc_thr = 0,
  p_col = "adj.P.Val",
  comparisons = NULL,
  assay = NULL,
  output_dir = NULL,
  verbose = TRUE,
  species_colors = NULL,
```

```

    species_df = NULL
  )

```

Arguments

<code>de_res</code>	Data frame with DE results. Required columns: Protein.IDs, logFC, P.Value, adj.P.Val, Change, Comparison. Optional: Gene.Names, Assay, Species (if not using <code>species_df</code>).
<code>expected_values</code>	Data frame with expected values. Required columns: Comparison, Species, expected_logFC.
<code>alpha</code>	Significance threshold (default: 0.05)
<code>lfc_thr</code>	Log fold-change threshold (default: 0)
<code>p_col</code>	P-value column to use (default: "adj.P.Val")
<code>comparisons</code>	Comparisons to include (NULL = all)
<code>assay</code>	Assay to filter (NULL = all)
<code>output_dir</code>	Directory for exported files (NULL = no export)
<code>verbose</code>	Print progress messages (default: TRUE)
<code>species_colors</code>	Named vector of colors per species (optional)
<code>species_df</code>	Data frame with Protein.IDs and Species columns (optional). If NULL, <code>de_res</code> must already contain a Species column.

Value

List with:

- `metrics_table`: Classification metrics per comparison
- `confusion_by_species`: Confusion matrix per Comparison x Species
- `confusion_overall`: Confusion matrix aggregated per Comparison
- `dispersion_metrics`: Dispersion stats per Comparison x Species
- `classified_df`: Full classified data frame
- `gg_heatmap`: ggplot2 performance heatmap
- `gg_confusion_by_species`: ggplot2 confusion matrix heatmap (by species)
- `gg_confusion_overall`: ggplot2 confusion matrix heatmap (aggregated)
- `gg_aucBars`: ggplot2 AUC bar chart
- `gg_metrics_bars`: ggplot2 grouped metrics bar chart
- `gg_signif_bars`: ggplot2 significant proteins stacked bars (absolute)
- `hc_volcano_list`: Named list of Highcharter volcano plots
- `parameters`: List of parameters used

Examples

```

data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
species_df <- data.frame(Protein.IDs = sp$PG.ProteinGroups,
                        Species = sp$PG.OrganismId)
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                Species = c("ECOLI", "YEAST"),
                expected_logFC = c(1, -0.58, 2, -3.3))
bench <- benchmarking_proteomics(de, ev, species_df = species_df,
                                verbose = FALSE)

bench$metrics_table
bench$gg_heatmap

```

bm_compute_extended_ranking

Compute extended ranking across 11 classification + OpDEA metrics

Description

For each of the 11 metrics, averages across comparisons per Assay (mean), then ranks methods in descending order (higher = better for all metrics). Final rank = mean of the 11 individual ranks.

Usage

```

bm_compute_extended_ranking(
  opdea_combined,
  bench_metrics_combined,
  metrics = .BM_EXTENDED_METRICS
)

```

Arguments

```

opdea_combined  data.frame from import_opdea_results()
bench_metrics_combined
                data.frame from import_benchmark_metrics()
metrics         Character vector of metrics to include (default: all 11 extended metrics)

```

Details

Note: this extended ranking can drift away from the canonical OpDEA ranking (bm_compute_ranking, based on nMCC/G_mean/pAUC). Performance is collinear with F1 (so F1 effectively counts twice), and in spike-in designs Accuracy/NPV are dominated by the background (TN »), hence they have little discriminative power. Use it as a complementary view, not as a substitute for the OpDEA ranking.

Value

Named list with:

extended_combined Merged data.frame (long format)
mean_aggregated data.frame of mean metric values per Assay
mean_ranking data.frame with rank columns and rank_final
n_methods Number of unique methods
n_comparisons Number of unique comparisons
metrics_used Metrics actually used

Examples

```
data(nadiaDia)
sp <- utils::read.delim(system.file("extdata", "nadiaDia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- NULL
bench_all <- NULL
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadiaDia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  opdea_all <- rbind(opdea_all, cbind(compute_opdea_metrics(de, ev), Assay = nm))
  bench_all <- rbind(bench_all, cbind(compute_benchmark_metrics(de, ev), Assay = nm))
}
ext <- bm_compute_extended_ranking(opdea_all, bench_all)
ext$mean_ranking[, c("Assay", "rank_final")]
```

bm_compute_ranking	<i>Compute OpDEA ranking across multiple methods</i>
--------------------	--

Description

For each metric, aggregates across comparisons (mean and median), then ranks methods in descending order (rank 1 = best). Final rank = average of individual metric ranks.

Usage

```
bm_compute_ranking(opdea_combined, metrics = .BM_METRICS)
```

Arguments

opdea_combined data.frame with columns: Assay, Comparison, nMCC, G_mean, pAUC_001, pAUC_005, pAUC_010
metrics Character vector of metrics to include in ranking (default: all 5 OpDEA metrics)

Value

Named list with:

mean_aggregated data.frame of mean metric values per Assay
median_aggregated data.frame of median metric values per Assay
mean_ranking data.frame of ranks based on mean aggregation
median_ranking data.frame of ranks based on median aggregation
opdea_combined Input combined data.frame
n_methods Number of methods
n_comparisons Number of comparisons
metrics_used Metrics included in ranking

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
})))
ranking <- bm_compute_ranking(opdea_all)
ranking$mean_ranking
```

bm_compute_ranking_by_comparison

Compute OpDEA ranking separately for each comparison

Description

For each comparison, ranks methods directly on their metric values (no aggregation needed since there is one value per method per comparison).

Usage

```
bm_compute_ranking_by_comparison(opdea_combined, metrics = .BM_METRICS)
```

Arguments

opdea_combined data.frame with columns: Assay, Comparison, nMCC, G_mean, pAUC_001, pAUC_005, pAUC_010
metrics Character vector of metrics to include in ranking

Value

Named list with:

by_comparison Named list where each key is a comparison and each value is a data.frame with metric values, ranks, and rank_final

ranking_combined data.frame with all comparisons in long format (includes Comparison column)

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
by_comp <- bm_compute_ranking_by_comparison(opdea_all)
by_comp$by_comparison[["B-A"]]
```

bm_plot_confusion_stacked

Stacked bar chart of TP/FP/FN/TN (OpDEA Figure 5 style)

Description

Horizontal stacked bars showing confusion matrix counts per method. If comparison is NULL, generates a faceted plot with all comparisons.

Usage

```
bm_plot_confusion_stacked(confusion_combined, comparison = NULL, title = NULL)
```

Arguments

confusion_combined	data.frame with columns: Assay, Comparison, TP, FP, TN, FN
comparison	Optional: single comparison to plot. If NULL, faceted plot with all comparisons.
title	Optional plot title

Value

ggplot2 object

Examples

```

data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
bench_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_benchmark_metrics(de, ev), Assay = nm)
}))
bm_plot_confusion_stacked(bench_all)
bm_plot_confusion_stacked(bench_all, comparison = "B-A")

```

bm_plot_extended_ranking_bars

Horizontal bar chart of extended combined ranking

Description

Horizontal bar chart of extended combined ranking

Usage

```
bm_plot_extended_ranking_bars(extended_ranking, title = NULL)
```

Arguments

extended_ranking
List from bm_compute_extended_ranking()

title
Optional plot title

Value

ggplot object

Examples

```

data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- NULL
bench_all <- NULL
for (nm in c("cycloess", "quantile")) {

```

```

de <- process_proteomics(nadia_dia, norm_method = nm,
                        verbose = FALSE)$DEPs_results
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
opdea_all <- rbind(opdea_all, cbind(compute_opdea_metrics(de, ev), Assay = nm))
bench_all <- rbind(bench_all, cbind(compute_benchmark_metrics(de, ev), Assay = nm))
}
ext <- bm_compute_extended_ranking(opdea_all, bench_all)
bm_plot_extended_rankingBars(ext)

```

bm_plot_extended_ranking_heatmap

Heatmap of extended ranking (11 metrics)

Description

Heatmap of extended ranking (11 metrics)

Usage

```
bm_plot_extended_ranking_heatmap(extended_ranking, title = NULL)
```

Arguments

extended_ranking	
title	List from bm_compute_extended_ranking() Optional plot title

Value

ggplot object

Examples

```

data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- NULL
bench_all <- NULL
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                          verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  opdea_all <- rbind(opdea_all, cbind(compute_opdea_metrics(de, ev), Assay = nm))
  bench_all <- rbind(bench_all, cbind(compute_benchmark_metrics(de, ev), Assay = nm))
}
ext <- bm_compute_extended_ranking(opdea_all, bench_all)
bm_plot_extended_ranking_heatmap(ext)

```

bm_plot_metrics_comparison

Boxplot comparison of metric distributions across methods

Description

Faceted boxplot showing the distribution of each metric across comparisons for every method.

Usage

```
bm_plot_metrics_comparison(opdea_combined, metrics = .BM_METRICS, title = NULL)
```

Arguments

opdea_combined	data.frame with Assay, Comparison, and metric columns
metrics	Character vector of metrics to plot (default: all 5)
title	Optional plot title

Value

ggplot2 object

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
bm_plot_metrics_comparison(opdea_all)
```

bm_plot_metrics_heatmap

Heatmap of aggregated metric values (not ranks)

Description

Shows the actual metric values per method, with per-column color scaling.

Usage

```
bm_plot_metrics_heatmap(ranking_result, type = "mean", title = NULL)
```

Arguments

ranking_result	List returned by <code>bm_compute_ranking()</code>
type	"mean" or "median" aggregation (default: "mean")
title	Optional plot title

Value

ggplot2 object

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
ranking <- bm_compute_ranking(opdea_all)
bm_plot_metrics_heatmap(ranking)
```

bm_plot_ranking_bars *Bar chart of final ranks*

Description

Horizontal bars of rank_final by Assay, ordered best to worst.

Usage

```
bm_plot_ranking_bars(ranking_result, type = "mean", title = NULL)
```

Arguments

ranking_result List returned by bm_compute_ranking()
 type "mean" or "median" aggregation (default: "mean")
 title Optional plot title

Value

ggplot2 object

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
ranking <- bm_compute_ranking(opdea_all)
bm_plot_ranking_bars(ranking)
```

bm_plot_ranking_bars_by_comp
 Bar chart of final ranks for a single comparison

Description

Bar chart of final ranks for a single comparison

Usage

```
bm_plot_rankingBars_by_comp(ranking_by_comp, comparison, title = NULL)
```

Arguments

`ranking_by_comp` List returned by `bm_compute_ranking_by_comparison()`

`comparison` Character string: which comparison to plot

`title` Optional plot title

Value

ggplot2 object

Examples

```
data(nadiaDia)
sp <- utils::read.delim(system.file("extdata", "nadiaDia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadiaDia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
by_comp <- bm_compute_ranking_by_comparison(opdea_all)
bm_plot_rankingBars_by_comp(by_comp, comparison = "D-A")
```

`bm_plot_ranking_heatmap`

Heatmap of metric ranks by method

Description

Rows = Assay (ordered by `rank_final`), columns = metrics. Fill color = rank value (green = best, red = worst).

Usage

```
bm_plot_ranking_heatmap(ranking_result, type = "mean", title = NULL)
```

Arguments

`ranking_result` List returned by `bm_compute_ranking()`

`type` "mean" or "median" aggregation (default: "mean")

`title` Optional plot title

Value

ggplot2 object

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
ranking <- bm_compute_ranking(opdea_all)
bm_plot_ranking_heatmap(ranking)
```

bm_plot_ranking_heatmap_by_comp

Heatmap of metric ranks for a single comparison

Description

Heatmap of metric ranks for a single comparison

Usage

```
bm_plot_ranking_heatmap_by_comp(ranking_by_comp, comparison, title = NULL)
```

Arguments

ranking_by_comp	List returned by <code>bm_compute_ranking_by_comparison()</code>
comparison	Character string: which comparison to plot
title	Optional plot title

Value

ggplot2 object

Examples

```

data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
opdea_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cbind(compute_opdea_metrics(de, ev), Assay = nm)
}))
by_comp <- bm_compute_ranking_by_comparison(opdea_all)
bm_plot_ranking_heatmap_by_comp(by_comp, comparison = "B-A")

```

bm_plot_roc

*ROC Curves by Method for a Single Comparison***Description**

Generates ROC (or ROC zoom) plot where each curve represents a different method (Assay) for one comparison.

Usage

```

bm_plot_roc(
  classified_combined,
  comparison,
  p_col = "adj.P.Val",
  zoom = FALSE,
  palette = "Set2"
)

```

Arguments

classified_combined	data.frame with columns: Assay, Comparison, truth, and a p-value column
comparison	Character. Single comparison to plot (e.g. "B-A")
p_col	P-value column name (default: "adj.P.Val")
zoom	Logical. If TRUE, zoom to FPR 0-10% and show pAUC in legend
palette	RColorBrewer palette name (default: "Set2")

Value

ggplot2 object or NULL if pROC not available

Examples

```

data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
classified_all <- do.call(rbind, lapply(c("cycloess", "quantile"), function(nm) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  cl <- benchmarking_proteomics(de, ev, verbose = FALSE)$classified_df
  cl$Assay <- nm
  cl
}))
if (requireNamespace("pROC", quietly = TRUE)) {
  bm_plot_roc(classified_all, comparison = "B-A")
}

```

boxplot_highchart_list

Interactive Highcharts Boxplot for Proteomics

Description

Interactive Highcharts Boxplot for Proteomics

Usage

```

boxplot_highchart_list(
  data,
  assays = NULL,
  color_by = "Condition",
  group_order = NULL,
  palette = NULL,
  title = NULL,
  subtitle = NULL,
  show_outliers = TRUE,
  outlier_jitter = 0.15,
  outlier_size = 3,
  box_width = 20,
  horizontal = TRUE,
  height = NULL
)

```

Arguments

data	Data frame with columns: Column, Assay, Intensity, Condition
assays	Vector of assays to include (NULL = all of them)
color_by	Column used for colouring (default: "Condition")
group_order	Order of the groups/conditions
palette	Colour palette: "ggsci::palette", "brewer:Name", or a vector
title	Chart title (optional). Use {assay} as a placeholder (e.g. "Boxplot: {assay}" -> "Boxplot: ImpSeqRob_Min")
subtitle	Chart subtitle (optional). Use {assay} as a placeholder
show_outliers	Show outliers lying outside the whiskers (default: TRUE)
outlier_jitter	Amount of horizontal jitter applied to the outliers (default: 0.15)
outlier_size	Radius of the outlier points (default: 3)
box_width	Width of the boxplot boxes in pixels (default: 20)
horizontal	Horizontal orientation (default: TRUE)
height	Chart height in pixels

Value

List of highchart objects (one per assay)

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# One boxplot per assay; keep only the imputed one
hc_list <- boxplot_highchart_list(
  res$BoxPlot_Input,
  assays      = "Impseqrob_min",
  group_order = c("A", "B", "D"),
  title       = "Boxplot: {assay}"
)
names(hc_list)
```

build_clustering_matrix

Build the clustering matrix aggregated by condition

Description

Build the clustering matrix aggregated by condition

Usage

```
build_clustering_matrix(
  intensity_matrix,
  sample_metadata,
  selected_features,
  condition_order = NULL,
  aggregate = c("median", "mean"),
  verbose = TRUE
)
```

Arguments

intensity_matrix	Intensity matrix (features x samples)
sample_metadata	DataFrame with SampleID, Condition
selected_features	Vector of features to include
condition_order	Condition order (NULL = alphabetical order)
aggregate	Aggregation method: "median" or "mean"
verbose	Print progress messages (default: TRUE)

Value

Matrix of aggregated intensities (features x conditions)

build_long_output	<i>Build the output table in LONG format</i>
-------------------	--

Description

Creates one row per FeatureID-Cluster combination where membership $\geq$ min_membership. This lets a protein appear in several clusters (soft clustering).

Usage

```
build_long_output(cl, eset_std, conditions, min_membership)
```

Arguments

cl	Mfuzz clustering object
eset_std	Standardised ExpressionSet (with z-scores)
conditions	Vector of condition names
min_membership	Minimum membership threshold

Value

DataFrame in long format

build_pca_scores	<i>Build a PCA scores data frame from data in long format</i>
------------------	---

Description

Build a PCA scores data frame from data in long format

Usage

```
build_pca_scores(
  pca_input,
  mode = c("all", "any", "specific"),
  alpha = 0.05,
  comparison = NULL,
  subset_label = NULL,
  center = TRUE,
  scale. = TRUE,
  filter_samples_to_comparison = FALSE,
  cond_col = "Condition"
)
```

Arguments

pca_input	Data frame in long format with columns: • SampleID: Sample identifier • FeatureID: Protein/feature identifier • Intensity: Intensity value (log2) • Condition: Experimental condition • Replicate: Replicate number (optional) • sig_any: Logical flagging significance in any comparison (for mode="any") • adjP_*: Adjusted p-value columns, one per comparison (for mode="specific")
mode	Protein filtering mode: "all", "any", or "specific"
alpha	Significance threshold for mode "specific" (default: 0.05)
comparison	Name of the comparison for mode "specific" (e.g. "B-A")
subset_label	Custom label for the subset (optional)
center	Center the data before PCA (default: TRUE)
scale.	Scale the data before PCA (default: TRUE)
filter_samples_to_comparison	Restrict samples to the conditions involved in the specific comparison (default: FALSE)
cond_col	Name of the condition column (default: "Condition")

Value

Data frame with columns: SampleID, PC1, PC2, PC1_Perc, PC2_Perc, Subset, Condition, Replicate

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# Scores and the variance explained by the first two components
scores <- build_pca_scores(res$PCA_Input, mode = "all")
head(scores)
unique(scores[c("PC1_Perc", "PC2_Perc")])
```

```
cluster_centroids_highchart
```

Centroid plot for all the clusters

Description

Builds a comparative plot with the centroids of all the clusters.

Usage

```
cluster_centroids_highchart(
  data,
  conditions = NULL,
  clusters = NULL,
  min_membership = NULL,
  centroid_summary = c("mean", "median"),
  palette = NULL,
  line_width = 2.5,
  show_markers = TRUE,
  title = NULL,
  height = NULL
)
```

Arguments

data	Pattern Profiler DataFrame (LONG format)
conditions	Vector of condition names
clusters	Clusters to include (NULL = all)
min_membership	Membership filter applied before computing the centroids
centroid_summary	Method: "mean" or "median"

palette	Colour palette
line_width	Line width (default: 2.5)
show_markers	Show markers on the points (default: TRUE)
title	Custom title
height	Plot height

Value

highchart object

Examples

```
if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results,
                                assay_name = "Impseqrob_min",
                                auto_select_c = FALSE, c = 3,
                                verbose = FALSE)

  # All the cluster centroids on one chart
  hc <- cluster_centroids_highchart(pp$long_output,
                                   conditions = c("A", "B", "D"),
                                   centroid_summary = "median")

  print(class(hc))
}
```

cluster_profile_highchart

Cluster profile plot with Highcharts

Description

Builds an interactive plot showing the expression profiles of the proteins in a given cluster.

Usage

```
cluster_profile_highchart(
  data,
  cluster,
  conditions = NULL,
  min_membership = NULL,
  show_centroid = TRUE,
  centroid_summary = c("mean", "median"),
  cluster_color = NULL,
```

```

    line_width = 1,
    line_opacity = 0.4,
    centroid_width = 3,
    title = NULL,
    height = NULL
  )

```

Arguments

<code>data</code>	Pattern Profiler DataFrame (LONG format)
<code>cluster</code>	Number of the cluster to plot
<code>conditions</code>	Vector of condition names (order for the X axis)
<code>min_membership</code>	Additional membership filter (NULL = no filter)
<code>show_centroid</code>	Show the centroid line (default: TRUE)
<code>centroid_summary</code>	Method for the centroid: "mean" or "median"
<code>cluster_color</code>	Cluster colour (NULL = automatic)
<code>line_width</code>	Width of the profile lines (default: 1)
<code>line_opacity</code>	Opacity of the lines (default: 0.4)
<code>centroid_width</code>	Width of the centroid line (default: 3)
<code>title</code>	Custom title (optional)
<code>height</code>	Plot height in pixels

Value

highchart object

Examples

```

if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results,
                                assay_name = "Impseqrob_min",
                                auto_select_c = FALSE, c = 3,
                                verbose = FALSE)

  hc <- cluster_profile_highchart(pp$long_output, cluster = 1,
                                conditions = c("A", "B", "D"))
  print(class(hc))
}

```

cluster_profile_highchart_list

List of cluster profile plots with Highcharts

Description

Builds a list of interactive plots for all the clusters.

Usage

```
cluster_profile_highchart_list(
  data,
  conditions = NULL,
  clusters = NULL,
  min_membership = NULL,
  show_centroid = TRUE,
  centroid_summary = c("mean", "median"),
  palette = NULL,
  line_width = 1,
  line_opacity = 0.4,
  centroid_width = 3,
  height = NULL
)
```

Arguments

data	Pattern Profiler DataFrame (LONG format)
conditions	Vector of condition names
clusters	Vector of clusters to plot (NULL = all)
min_membership	Minimum membership filter
show_centroid	Show the centroid line (default: TRUE)
centroid_summary	Method for the centroid: "mean" or "median"
palette	Palette for the cluster colours
line_width	Width of the profile lines (default: 1)
line_opacity	Opacity of the lines (default: 0.4)
centroid_width	Width of the centroid line (default: 3)
height	Height of each plot in pixels

Value

Named list of highchart objects

Examples

```

if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results,
                                assay_name = "Impseqrob_min",
                                auto_select_c = FALSE, c = 3,
                                verbose = FALSE)

  hc_profiles <- cluster_profile_highchart_list(pp$long_output,
                                                conditions = c("A", "B", "D"))

  print(names(hc_profiles))
}

```

compute_benchmark_metrics

Compute benchmark metrics for all comparisons

Description

Compute benchmark metrics for all comparisons

Usage

```

compute_benchmark_metrics(
  de_res,
  ev,
  alpha = 0.05,
  lfc_thr = 0,
  p_col = "adj.P.Val",
  comparisons = NULL,
  assay = NULL,
  species_df = NULL
)

```

Arguments

de_res	Data frame with DE results
ev	Data frame with expected values
alpha	Significance threshold (default: 0.05)
lfc_thr	Log fold-change threshold (default: 0)
p_col	P-value column name (default: "adj.P.Val")
comparisons	Comparisons to include (NULL = all)
assay	Assay to filter (NULL = all)
species_df	Data frame with Protein.IDs and Species columns (optional). If NULL, de_res must already contain a Species column.

Value

Data frame with metrics per comparison

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
metrics <- compute_benchmark_metrics(de, ev)
metrics[, c("Comparison", "TP", "FP", "Sensitivity", "Specificity", "F1")]
```

compute_confidence_ellipse

Compute the confidence ellipse per group

Description

Computes the coordinates of a confidence ellipse based on the chi-squared distribution, similar to FactoMineR::coord.ellipse and ggplot2::stat_ellipse.

Usage

```
compute_confidence_ellipse(
  scores_df,
  group_col = "Condition",
  level = 0.95,
  npoints = 100
)
```

Arguments

scores_df	Data frame with columns PC1, PC2 and the grouping column
group_col	Name of the grouping column (default: "Condition")
level	Confidence level (default: 0.95)
npoints	Number of points used to draw the ellipse (default: 100)

Value

List of data frames, each one with columns: group, x, y

`compute_dispersion_metrics`*Compute dispersion metrics by Comparison x Species*

Description

Calculates MED, SD, CV, MAD, RCV, IQR, trimmed SD/CV for significant proteins with correct direction.

Usage

```
compute_dispersion_metrics(  
  de_res,  
  ev,  
  alpha = 0.05,  
  lfc_thr = 0,  
  p_col = "adj.P.Val",  
  comparisons = NULL,  
  assay = NULL,  
  trim = 0.1,  
  species_df = NULL  
)
```

Arguments

<code>de_res</code>	Data frame with DE results
<code>ev</code>	Data frame with expected values
<code>alpha</code>	Significance threshold (default: 0.05)
<code>lfc_thr</code>	Log fold-change threshold (default: 0)
<code>p_col</code>	P-value column name (default: "adj.P.Val")
<code>comparisons</code>	Comparisons to include (NULL = all)
<code>assay</code>	Assay to filter (NULL = all)
<code>trim</code>	Trimming proportion for trimmed SD/CV (default: 0.1)
<code>species_df</code>	Data frame with Protein.IDs and Species columns (optional). If NULL, <code>de_res</code> must already contain a Species column.

Value

Data frame with dispersion metrics

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
disp <- compute_dispersion_metrics(de, ev)
disp[, c("Comparison", "Species", "expected_logFC", "MED", "MAD", "RCV")]
```

compute_hulls	<i>Compute the convex hull (enclosing polygon) per group</i>
---------------	--

Description

Compute the convex hull (enclosing polygon) per group

Usage

```
compute_hulls(scores_df, group_col = "Condition")
```

Arguments

scores_df	Data frame with columns PC1, PC2 and the grouping column
group_col	Name of the grouping column (default: "Condition")

Value

List of data frames, each one with columns: group, x, y

compute_opdea_metrics	<i>Compute OpDEA metrics for all comparisons</i>
-----------------------	--

Description

Calculates additional benchmark metrics based on Peng et al. 2024 (Nature Comms, OpDEA framework):

- pAUC(0.01), pAUC(0.05), pAUC(0.1): Partial AUC with McClish correction
- nMCC: Normalized MCC = (MCC + 1) / 2
- G-mean: sqrt(Sensitivity * Specificity)

Usage

```
compute_opdea_metrics(
  de_res,
  ev,
  alpha = 0.05,
  lfc_thr = 0,
  p_col = "adj.P.Val",
  comparisons = NULL,
  assay = NULL,
  species_df = NULL
)
```

Arguments

de_res	Data frame with DE results
ev	Data frame with expected values
alpha	Significance threshold (default: 0.05)
lfc_thr	Log fold-change threshold (default: 0)
p_col	P-value column name (default: "adj.P.Val")
comparisons	Comparisons to include (NULL = all)
assay	Assay to filter (NULL = all)
species_df	Data frame with Protein.IDs and Species columns (optional). If NULL, de_res must already contain a Species column.

Value

Data frame with columns: Comparison, nMCC, G_mean, pAUC_001, pAUC_005, pAUC_010

Examples

```
data(nadia_dia)
de <- process_proteomics(nadia_dia, verbose = FALSE)$DEPs_results
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
compute_opdea_metrics(de, ev)
```

`configure_cluster_palette`*Configure the colour palette for the clusters*

Description

Configure the colour palette for the clusters

Usage

```
configure_cluster_palette(n_clusters, palette = NULL)
```

Arguments

<code>n_clusters</code>	Number of clusters
<code>palette</code>	Palette to use (NULL, "ggsci::name", "brewer:name")

Value

Vector of colours named by cluster

`create_expression_set` *Create an ExpressionSet for Mfuzz*

Description

Create an ExpressionSet for Mfuzz

Usage

```
create_expression_set(mat, feature_info = NULL)
```

Arguments

<code>mat</code>	Expression matrix (features x conditions)
<code>feature_info</code>	DataFrame with feature information (optional)

Value

ExpressionSet object

de_analysis_proteomics

Perform differential expression analysis on proteomics data

Description

Coordinates comparison specification, limma analysis, and result extraction.

Usage

```
de_analysis_proteomics(
  se,
  assay_name = NULL,
  comparisons = NULL,
  control = NULL,
  logFC_threshold = 0,
  alpha = 0.05,
  p_adj = TRUE,
  eBayes_trend = NULL,
  eBayes_robust = NULL,
  de_method = "limma",
  covariate_column = NULL,
  bio_replicate_column = NULL,
  condition_column = "Condition",
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment with imputed assay (output of impute_proteomics)
assay_name	Assay name to use. If NULL, uses the last available assay
comparisons	Comparisons to perform. If NULL, auto-generates (pairwise or vs control)
control	Control condition. If NULL, generates all pairwise comparisons
logFC_threshold	LogFC threshold for significance (default: 0)
alpha	Adjusted p-value threshold (default: 0.05)
p_adj	Use adjusted p-value (default: TRUE)
eBayes_trend	Use trend estimation in eBayes (default: TRUE, recommended for proteomics)
eBayes_robust	Use robust estimation in eBayes (default: TRUE, recommended for proteomics)
de_method	DE method: "limma" (default) or "limpa" (probabilistic, requires imp_method="limpa")
covariate_column	Column name(s) in colData for paired/blocked design. Single string (e.g., "Subject") or character vector (e.g., c("Subject", "Batch")). Default: NULL

bio_replicate_column
Column name in colData identifying biological replicates (e.g., "Patient", "Subject"). Used with `limma::duplicateCorrelation()` to account for technical replicates or paired designs via random effect blocking. Default: NULL

condition_column
Condition column name (default: "Condition")

verbose
Print progress messages (default: TRUE)

Value

List with:

- **DEPs_results**: Data frame with differential expression results
- **comparisons**: Comparisons performed
- **assay_name**: Assay name used

Examples

```
data(nadia_dia)
norm <- normalize_proteomics(nadia_dia, norm_method = "cycloess",
                             verbose = FALSE)
imp  <- impute_proteomics(norm$se, normalized_assay_name = "cycloess",
                           verbose = FALSE)

# All pairwise comparisons
de <- de_analysis_proteomics(imp$se, assay_name = "Impseqrob_min",
                             verbose = FALSE)
head(de$DEPs_results[, c("Protein.IDs", "Comparison", "logFC", "adj.P.Val")])

# Or every condition against a single control
de2 <- de_analysis_proteomics(imp$se, assay_name = "Impseqrob_min",
                              control = "A", verbose = FALSE)
unique(de2$DEPs_results$Comparison)
```

deps_with_clusters	<i>Differential abundance with the Pattern Profiler cluster of each protein</i>
--------------------	---

Description

Joins the differential-abundance results to the soft clustering, giving the table a functional-enrichment or over-representation analysis needs: the statistics that say which proteins moved, and the cluster that says which pattern they follow.

Usage

```
deps_with_clusters(
  DEPs_results,
  pattern_profiler,
  assignment = c("primary", "all"),
  min_membership = NULL,
  comparison = NULL,
  significant_only = FALSE
)
```

Arguments

DEPs_results	Differential-abundance results, the DEPs_results element of <code>process_proteomics()</code> . Needs at least Protein.IDs and Comparison.
pattern_profiler	Result of <code>pattern_profiler_analysis()</code> , or NULL for no clustering at all, in which case the three cluster columns are returned as NA and the shape of the table is unchanged.
assignment	"primary" (default) for one row per protein and comparison, carrying the cluster of highest membership; "all" for one row per protein, comparison and cluster.
min_membership	Optional extra membership threshold. It can only be stricter than the one the analysis ran with: pairs below that one were discarded when long_output was built and cannot be recovered here. A protein left with no cluster keeps its row, with NA.
comparison	Optional character vector of comparisons to keep. The default, NULL, keeps them all.
significant_only	Keep only the rows classified Up or Down. The classification is the one the analysis made, with its own alpha and logFC_threshold; nothing is recomputed here.

Details

The join is a left join from the results, so **every protein that was tested survives it**, with or without a cluster. That matters more than it looks: the complete set of tested proteins is the background an enrichment is computed against, and a protein with no cluster – one that was never selected for the clustering, or that the clustering dropped – belongs in that background rather than in the bin.

Clustering is fuzzy, so a protein can belong to several clusters at once. assignment decides what to do about it. "primary" keeps only the dominant pattern of each protein and returns exactly one row per protein and comparison, which is what most enrichment tools expect. "all" keeps every association above the threshold, which is the honest representation of a soft clustering and the one to use when the analysis downstream can weight by membership.

Value

A data frame with the columns of DEPs_results followed by Cluster, Membership and ClusterRank (1 is the protein's dominant cluster).

See Also

[nadia_deps_with_clusters\(\)](#) for the same table read from a .nadia file, and [pattern_profiler_analysis\(\)](#) for the clustering.

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# Without a clustering the shape is the same and the clusters are NA.
fi <- deps_with_clusters(res$DEPs_results, NULL)
head(fi)

if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("Biobase", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results,
                                assay_name = "Impseqrob_min",
                                verbose = FALSE)

  primary <- deps_with_clusters(res$DEPs_results, pp)
  nrow(primary) == nrow(res$DEPs_results) # the background is intact

  soft <- deps_with_clusters(res$DEPs_results, pp, assignment = "all")
  table(soft$ClusterRank)
}
```

detect_condition_columns

Detect the condition columns (z-scores)

Description

Detect the condition columns (z-scores)

Usage

```
detect_condition_columns(data)
```

Arguments

data Pattern Profiler DataFrame

Value

Vector with the names of the condition columns

`evaluate_cluster_range`*Evaluate a range of cluster numbers*

Description

Evaluate a range of cluster numbers

Usage

```
evaluate_cluster_range(  
  eset_std,  
  c_range = 2:10,  
  m = NULL,  
  seeds = c(42, 123, 456),  
  verbose = TRUE  
)
```

Arguments

<code>eset_std</code>	Standardised ExpressionSet
<code>c_range</code>	Vector of cluster numbers to evaluate
<code>m</code>	Fuzziness parameter
<code>seeds</code>	Seeds for reproducibility
<code>verbose</code>	Show progress

Value

DataFrame with the metrics per number of clusters

`extract_se_data`*Extract data from a SummarizedExperiment*

Description

Retrieves the intensity matrix, the feature IDs and the sample metadata from a SummarizedExperiment object.

Usage

```
extract_se_data(se_proc, assay_name = NULL, verbose = TRUE)
```

Arguments

se_proc	SummarizedExperiment object
assay_name	Name of the assay to use (NULL = first available)
verbose	Print progress messages (default: TRUE)

Value

List with:

- intensity_matrix: features x samples matrix
- feature_ids: vector of protein IDs
- sample_metadata: DataFrame with SampleID, Condition, etc.

filter_significant_features
Filter features by significance

Description

Filter features by significance

Usage

```
filter_significant_features(
  feature_info,
  filter_mode = c("any", "all", "specific"),
  comparison = NULL,
  alpha = 0.05,
  verbose = TRUE
)
```

Arguments

feature_info	DataFrame with significance columns
filter_mode	Selection mode: • "any": features significant in AT LEAST one comparison (uses sig_any). • "all": ALL features, with no significance filtering (it does NOT mean "significant in every comparison"). • "specific": significant in the comparison given by comparison.
comparison	Specific comparison (for mode="specific")
alpha	Significance threshold
verbose	Print progress messages (default: TRUE)

Value

Vector of the selected FeatureIDs

`get_annotation_palette`*Build the Colour Palette for Categorical Annotations*

Description

Build the Colour Palette for Categorical Annotations

Usage

```
get_annotation_palette(levels, palette = NULL)
```

Arguments

levels	Vector of levels (categories)
palette	Palette specification (same format as <code>get_heatmap_palette</code>)

Value

Named vector of colours

`get_heatmap_palette`*Build the Colour Palette for the Heatmap Values*

Description

Build the Colour Palette for the Heatmap Values

Usage

```
get_heatmap_palette(palette = NULL, n = 11, reverse = FALSE)
```

Arguments

palette	Palette specification: • NULL: use the default palette (RdBu) • Vector of colours: use those colours directly • String "paletteer::" (e.g. "viridis::viridis"): use paletteer • String "brewer:" (e.g. "brewer:RdBu"): use RColorBrewer
n	Number of colours to generate
reverse	Reverse the palette (default: FALSE)

Value

Vector of colours, or a `colorRamp2` function

im_compute_metrics	<i>Compute imputation quality metrics (NRMSE, SOR, PSS, ACC_OI)</i>
--------------------	---

Description

Extracts complete-case rows from the assay, introduces artificial NAs, re-imputes with each method, and computes four quality metrics plus a combined ranking.

Usage

```
im_compute_metrics(
  se,
  assay_name = NULL,
  methods = .IM_BENCH_METHODS,
  combo_methods = list(),
  condition_col = "Condition",
  na_prop = 0.2,
  seed = 42L,
  pattern = "random",
  method_args = list(),
  with_value = NA_real_,
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment (from import_imp_matrices() or pipeline)
assay_name	Character. Name of the assay to use as starting point (typically the normalized assay, pre-imputation, with real NAs).
methods	Character vector of individual imputation methods to benchmark. Default: .IM_BENCH_METHODS (16 methods). Use character(0) or NULL to skip individual methods.
combo_methods	Named list of combo/softHybrid configurations. Each element is a list with: mode ("combo" or "softHybrid", default "combo"), mar_method, mnar_method. The list name is used as the method label in results. Default: empty list (no combo methods).
condition_col	Character. Column in colData with condition labels, required for combo methods. Default "Condition".
na_prop	Numeric (0-1). Proportion of artificial NAs. Default 0.20. Only used when pattern = "random". Ignored when pattern = "from_data" (NA proportion is derived from the data, replicating NAguidesR).
seed	Integer. Random seed. Default 42.
pattern	Character. NA introduction pattern: "random" for uniform random NAs, or "from_data" to replicate NAguidesR's simulation strategy (row ratio + per-column distribution from the actual data).
method_args	Named list of per-method argument lists.

with_value Constant value for method "with".
 verbose Logical. Print progress. Default TRUE.

Value

data.frame with columns: Method, NRMSE, SOR, PSS, ACC_OI, NRMSE_Rank, SOR_Rank, PSS_Rank, ACC_OI_Rank, Rank_Mean. Ordered by Rank_Mean (best first).

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)

# Ground-truth simulation: complete rows are masked and re-imputed, so the
# error of each method can be measured against the values it did not see
metrics <- im_compute_metrics(se, methods = c("min", "zero", "knn"),
                             verbose = FALSE)
metrics[, c("Method", "NRMSE", "SOR", "Rank_Mean")]
```

im_plot_acc_oi	<i>ACC_OI bar plot per imputation method</i>
----------------	--

Description

Bar chart of Average Correlation Coefficient (ACC_OI), ordered descending (higher is better).

Usage

```
im_plot_acc_oi(metrics_df, ...)
```

Arguments

metrics_df data.frame from im_compute_metrics().
 ... Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)
metrics <- im_compute_metrics(se, methods = c("min", "zero", "knn"),
                             verbose = FALSE)
im_plot_acc_oi(metrics)
```

im_plot_metrics	<i>Faceted bar chart of all 4 imputation quality metrics</i>
-----------------	--

Description

Similar to nm_plot_metrics(): one facet per metric with free y-scales, labeled values on bars.

Usage

```
im_plot_metrics(metrics_df, ...)
```

Arguments

metrics_df	data.frame from im_compute_metrics().
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)
metrics <- im_compute_metrics(se, methods = c("min", "zero", "knn"),
                             verbose = FALSE)
im_plot_metrics(metrics)
```

im_plot_nrmse	<i>NRMSE bar plot per imputation method</i>
---------------	---

Description

Bar chart of NRMSE values, ordered ascending (lower is better).

Usage

```
im_plot_nrmse(metrics_df, ...)
```

Arguments

metrics_df	data.frame from im_compute_metrics().
...	Additional arguments (unused).

Value

ggplot object.

Examples

[illegible]im_plot_pss *PSS bar plot per imputation method*

Description

Bar chart of Procrustes Statistical Shape (PSS), ordered ascending (lower is better). Requires vegan; NA values shown as 0-height bars.

Usage

```
im_plot_pss(metrics_df, ...)
```

Arguments

<code>metrics_df</code>	data.frame from <code>im_compute_metrics()</code> .
<code>...</code>	Additional arguments (unused).

Value

ggplot object.

Examples

[illegible]

im_plot_ranking	<i>Ranking heatmap of imputation methods</i>
-----------------	--

Description

Heatmap (geom_tile + geom_text) showing per-metric ranks and the combined Rank_Mean. Rows = methods (ordered by Rank_Mean), columns = metrics.

Usage

```
im_plot_ranking(metrics_df, ...)
```

Arguments

metrics_df	data.frame from im_compute_metrics().
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)
metrics <- im_compute_metrics(se, methods = c("min", "zero", "knn"),
                             verbose = FALSE)
im_plot_ranking(metrics)
```

im_plot_sor	<i>SOR bar plot per imputation method</i>
-------------	---

Description

Bar chart of Sum of Ranks (SOR), ordered ascending (lower is better).

Usage

```
im_plot_sor(metrics_df, ...)
```

Arguments

metrics_df	data.frame from im_compute_metrics().
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)
metrics <- im_compute_metrics(se, methods = c("min", "zero", "knn"),
                             verbose = FALSE)
im_plot_sor(metrics)
```

im_prepare_se

Build a SummarizedExperiment for imputation benchmarking

Description

Convenience function that creates a SE from a proteomics_data object and applies the winning normalization method (from normalization benchmarking). The result is ready for imputation_metrics().

Usage

```
im_prepare_se(
  preprocessing,
  norm_method = NULL,
  pc1_rank = NULL,
  min_reps = NULL,
  min_groups = 1,
  covariate_df = NULL,
  norm_method_args = list(),
  include_log2 = TRUE,
  verbose = TRUE
)
```

Arguments

preprocessing	A proteomics_data object, from any of the preprocess_*() functions.
norm_method	Character scalar. Normalization method name (e.g. "cycloess"). If provided, used directly. Default NULL.
pc1_rank	data.frame from nm_rank_pc1() or normalization_metrics()\$pc1_rank. The first row's Method column is used as winner. Ignored if norm_method is provided. Default NULL.
min_reps	Minimum replicates with non-NA values per group for protein filtering. If NULL, auto-computed as half the smallest group. Default NULL.
min_groups	Minimum groups meeting min_reps (default: 1).
covariate_df	Optional covariate data.frame for paired designs (must contain a Column column). Default NULL.

norm_method_args Named list of per-method arguments for normalization. Default list().

include_log2 Logical. Include the baseline "log2" assay in the returned SE. Default TRUE.

verbose Logical. Print progress messages. Default TRUE.

Details

Internally calls nm_prepare_se() to build the baseline SE with assay "log2", then applies the chosen normalization via .nm_dispatch_normalization().

Value

SummarizedExperiment with assays "log2" (optional) + winner method.

Examples

```
data(nadia_dia)

# Option A: name the normalization to benchmark imputation on
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)
SummarizedExperiment::assayNames(se)

# Option B: let the winner of normalization_metrics() decide. That function
# takes a SummarizedExperiment, so the baseline is built first.
se_base <- nm_prepare_se(nadia_dia, verbose = FALSE)
nm <- normalization_metrics(se_base,
                           methods = c("cycloess", "quantile", "MAD"),
                           plots = "pc1_ranking", verbose = FALSE)

nm$pc1_rank
se2 <- im_prepare_se(nadia_dia, pc1_rank = nm$pc1_rank, verbose = FALSE)
SummarizedExperiment::assayNames(se2)
```

import_benchmark_metrics

Import benchmark metrics from multiple benchmark result folders

Description

Reads benchmark_metrics.tsv files from subdirectories, adding an Assay column derived from the folder name.

Usage

```
import_benchmark_metrics(
  results_dir,
  pattern = "benchmark_metrics\\.tsv$",
  method_names = NULL,
  recursive = TRUE
)
```

Arguments

results_dir	Parent directory containing benchmark subfolders
pattern	Regex pattern for the metrics filename
method_names	Optional character vector of Assay names
recursive	Search subdirectories recursively? (default: TRUE)

Value

data.frame with columns: Assay, Comparison, TP, FP, TN, FN, Sensitivity, Specificity, Precision, NPV, Accuracy, F1, MCC, AUC, Performance

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
root <- file.path(tempdir(), "nadia_bench")
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  benchmarking_proteomics(de, ev, output_dir = file.path(root, nm),
                           verbose = FALSE)
}
bench_all <- import_benchmark_metrics(root)
bench_all[, c("Assay", "Comparison", "Sensitivity", "Specificity", "F1")]
```

import_classified_results

Import classified results from multiple benchmark result folders

Description

Reads benchmark_classified.tsv files from subdirectories, adding an Assay column derived from the folder name.

Usage

```
import_classified_results(
  results_dir,
  pattern = "benchmark_classified\\.tsv$",
  method_names = NULL,
  recursive = TRUE
)
```

Arguments

results_dir	Parent directory containing benchmark subfolders
pattern	Regex pattern for the classified filename
method_names	Optional character vector of Assay names
recursive	Search subdirectories recursively? (default: TRUE)

Value

data.frame with columns from classified_df plus Assay

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
root <- file.path(tempdir(), "nadia_bench")
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  benchmarking_proteomics(de, ev, output_dir = file.path(root, nm),
                           verbose = FALSE)
}
classified_all <- import_classified_results(root)
table(classified_all$Assay, classified_all$classification)
```

import_confusion_results

Import confusion matrices from multiple benchmark result folders

Description

Reads benchmark_confusion_overall.tsv files from subdirectories, adding an Assay column derived from the folder name.

Usage

```
import_confusion_results(
  results_dir,
  pattern = "benchmark_confusion_overall\\.tsv$",
  method_names = NULL,
  recursive = TRUE
)
```

Arguments

results_dir	Parent directory containing benchmark subfolders
pattern	Regex pattern for the confusion filename
method_names	Optional character vector of Assay names
recursive	Search subdirectories recursively? (default: TRUE)

Value

data.frame with columns: Assay, Comparison, N, TP, FP, TN, FN, plus percentage columns

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
root <- file.path(tempdir(), "nadia_bench")
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                           verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  benchmarking_proteomics(de, ev, output_dir = file.path(root, nm),
                           verbose = FALSE)
}
confusion_all <- import_confusion_results(root)
confusion_all[, c("Assay", "Comparison", "TP", "FP", "TN", "FN")]
```

import_imp_matrices *Import imputed matrices into a SummarizedExperiment*

Description

Reads all TSV files matching a pattern from a directory, one assay per file. Each TSV must have column 1 as protein identifier and remaining columns as sample names matching the sample_col in the metadata file.

Usage

```
import_imp_matrices(
  tsv_dir,
  metadata_path,
  pattern = "matrix_log2_.*\\.tsv$",
  method_names = NULL,
  condition_col = "Condition",
  sample_col = "Column"
)
```

Arguments

tsv_dir	Character. Directory containing the TSV files.
metadata_path	Character. Path to a TSV or CSV file with at minimum columns sample_col and condition_col.
pattern	Character. Regex to filter files in tsv_dir. Default: "matrix_log2_.*\\.tsv\$".
method_names	Character vector. Assay names to assign (in order). NULL (default) = extracted from file names between matrix_log2_ and .tsv.
condition_col	Character. Column in metadata with condition/group labels.
sample_col	Character. Column in metadata with sample names.

Value

A SummarizedExperiment with one assay per TSV, colData from metadata, and rowData containing Protein.IDs.

Examples

```
# Round trip: write the imputed matrices out, then read them back in
data(nadia_dia)
res <- process_proteomics(nadia_dia, export_dir = tempdir(),
                          export_format = "tsv", verbose = FALSE)

meta <- file.path(tempdir(), "metadata.tsv")
utils::write.table(
  data.frame(Column = res$se_proc$Column, Condition = res$se_proc$Condition,
             Replicate = res$se_proc$Replicate),
  meta, sep = "\t", row.names = FALSE, quote = FALSE)

se <- import_imp_matrices(tsv_dir = tempdir(), metadata_path = meta)
SummarizedExperiment::assayNames(se)
```

import_norm_matrices *Import normalized matrices into a SummarizedExperiment*

Description

Reads all TSV files matching a pattern from a directory, one assay per file. Each TSV must have column 1 named ProteinGroups and remaining columns as sample names matching the sample_col in the metadata file.

Usage

```
import_norm_matrices(
  tsv_dir,
  metadata_path,
  pattern = "matrix_log2_.*\\.tsv$",
```

```

    method_names = NULL,
    condition_col = "Condition",
    sample_col = "Column"
  )

```

Arguments

<code>tsv_dir</code>	Character. Directory containing the normalized TSV files.
<code>metadata_path</code>	Character. Path to a TSV or CSV file with at minimum columns <code>sample_col</code> and <code>condition_col</code> .
<code>pattern</code>	Character. Regex to filter files in <code>tsv_dir</code> . Default: <code>"matrix_log2_.*\\.tsv\$"</code> .
<code>method_names</code>	Character vector. Assay names to assign (in order). NULL (default) = extracted from file names between <code>matrix_log2_</code> and <code>.tsv</code> .
<code>condition_col</code>	Character. Column in metadata with condition/group labels.
<code>sample_col</code>	Character. Column in metadata with sample names.

Value

A SummarizedExperiment with one assay per TSV, colData from metadata, and rowData containing Protein.IDs.

Examples

```

data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "MAD"), verbose = FALSE)

# Write the normalized assays as TSVs, then read them back into a new SE
tsv_dir <- file.path(tempdir(), "nm_matrices")
dir.create(tsv_dir, showWarnings = FALSE)
for (a in c("cycloess", "MAD")) {
  m <- SummarizedExperiment::assay(se, a)
  utils::write.table(
    data.frame(ProteinGroups = rownames(m), m, check.names = FALSE),
    file.path(tsv_dir, paste0("matrix_log2_", a, ".tsv")),
    sep = "\t", row.names = FALSE, quote = FALSE)
}
meta_path <- file.path(tsv_dir, "metadata.tsv")
utils::write.table(as.data.frame(SummarizedExperiment::colData(se)), meta_path,
  sep = "\t", row.names = FALSE, quote = FALSE)

se_nm <- import_norm_matrices(tsv_dir, meta_path)
SummarizedExperiment::assayNames(se_nm)

```

import_opdea_results *Import OpDEA metrics from multiple benchmark result folders*

Description

Reads benchmark_opdea_metrics.tsv files from subdirectories, adding an Assay column derived from the folder name.

Usage

```
import_opdea_results(
  results_dir,
  pattern = "benchmark_opdea_metrics\\.tsv$",
  method_names = NULL,
  recursive = TRUE
)
```

Arguments

results_dir	Parent directory containing benchmark subfolders
pattern	Regex pattern for the opdea metrics filename (default: "benchmark_opdea_metrics\\.tsv\$")
method_names	Optional character vector of Assay names. If NULL, names are derived from the parent folder of each file.
recursive	Logical. Search subdirectories recursively? (default: TRUE)

Value

data.frame in long format with columns: Assay, Comparison, nMCC, G_mean, pAUC_001, pAUC_005, pAUC_010

Examples

```
data(nadia_dia)
sp <- utils::read.delim(system.file("extdata", "nadia_dia_spikein.tsv.gz",
                                   package = "NADIA"))
ev <- data.frame(Comparison = rep(c("B-A", "D-A"), each = 2),
                 Species = c("ECOLI", "YEAST"),
                 expected_logFC = c(1, -0.58, 2, -3.3))
root <- file.path(tempdir(), "nadia_bench")
for (nm in c("cycloess", "quantile")) {
  de <- process_proteomics(nadia_dia, norm_method = nm,
                          verbose = FALSE)$DEPs_results
  de$Species <- sp$PG.OrganismId[match(de$Protein.IDs, sp$PG.ProteinGroups)]
  benchmarking_proteomics(de, ev, output_dir = file.path(root, nm),
                          verbose = FALSE)
}
opdea_all <- import_opdea_results(root)
head(opdea_all)
```

imputation_metrics *Generate imputation quality metric plots and ranking*

Description

Computes NRMSE, SOR, PSS, ACC_OI via ground-truth simulation, then generates up to 6 diagnostic plots. Each plot is wrapped in tryCatch() so that a failure in one does not abort all.

Usage

```
imputation_metrics(
  se,
  assay_name = NULL,
  methods = .IM_BENCH_METHODS,
  combo_methods = list(),
  condition_col = "Condition",
  na_prop = 0.2,
  seed = 42L,
  pattern = "random",
  method_args = list(),
  with_value = NA_real_,
  plots = "all",
  verbose = TRUE,
  output_dir = NULL,
  export_plots = TRUE,
  export_tables = TRUE,
  plot_width = 12,
  plot_height = 8,
  plot_dpi = 150
)
```

Arguments

se	SummarizedExperiment (from import_imp_matrices() or pipeline).
assay_name	Character. Assay name to use as starting point. NULL (default) = first assay.
methods	Character vector of individual methods to benchmark. Default: .IM_BENCH_METHODS (16 methods). Use NULL or character(0) to skip individual methods when only using combo_methods.
combo_methods	Named list of combo/softHybrid configurations. Each element: list(mar_method, mnar_method, mode). See im_compute_metrics() for details.
condition_col	Character. Column in colData for combo methods. Default "Condition".
na_prop	Numeric (0-1). Proportion of artificial NAs. Default 0.20. Only used when pattern = "random". Ignored when pattern = "from_data" (NA proportion is derived from the data, replicating NAGuideR).
seed	Integer. Random seed. Default 42.

pattern	Character. "random" or "from_data". Default "random".
method_args	Named list of per-method argument lists.
with_value	Constant for method "with".
plots	Character vector of plot names or "all" (default). Valid: "nrmse", "sor", "pss", "acc_oi", "ranking", "metrics".
verbose	Logical. Print progress. Default TRUE.
output_dir	Character. Path to export directory. If NULL (default), no files are exported. When set, tables (TSV) and/or plots (PNG) are saved to this directory (created if needed).
export_plots	Logical. Export plots as PNG when output_dir is set. Default TRUE.
export_tables	Logical. Export tables as TSV when output_dir is set. Default TRUE.
plot_width	Numeric. Width in inches for exported plots. Default 12.
plot_height	Numeric. Height in inches for exported plots. Default 8.
plot_dpi	Numeric. Resolution for exported plots. Default 150.

Value

Named list of ggplot objects (or NULL for failed plots), plus `metrics_table`: a data.frame from `im_compute_metrics()`.

Examples

```
data(nadia_dia)
se <- im_prepare_se(nadia_dia, norm_method = "cycloess", verbose = FALSE)

# The default sweeps 16 methods; a handful is enough to illustrate it
res <- imputation_metrics(se, methods = c("min", "zero", "knn"),
  plots = "ranking", verbose = FALSE)
res$metrics_table

# Two-stage combos can be compared alongside the individual methods
res2 <- imputation_metrics(se, methods = c("min", "knn"),
  combo_methods = list(
    "Impseq+min" = list(mar_method = "Impseq", mnar_method = "min")),
  plots = "ranking", verbose = FALSE)
res2$metrics_table$Method
```

impute_proteomics

Impute proteomics data

Description

Complete imputation pipeline with 20 methods. Four pathways:

Usage

```
impute_proteomics(
  se,
  normalized_assay_name = "cycloess",
  imputed_assay_name = NULL,
  imp_method = "combo",
  mar_method = "Impseqrob",
  mnar_method = "min",
  prop_na_mnar = 0.51,
  prop_present_mar = 0.5,
  min_present_mar = 1,
  require_n_conditions = 1,
  max_na_prop = NULL,
  method_args = list(),
  with_value = NA_real_,
  verbose = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment with normalized assay
<code>normalized_assay_name</code>	Name of the normalized assay to use (default: "cycloess")
<code>imputed_assay_name</code>	Name for the imputed assay. If NULL, it is built from the method: <code>combo</code> gives <code><mar_method>_<mnar_method></code> , <code>softHybrid</code> gives <code>softHybrid_<mar_method>_<mnar_method></code> , and any single method gives its own name.
<code>imp_method</code>	Imputation method (default: "combo"). One of: "combo", "softHybrid", "bpca", "knn", "mice", "missForest", "Impseq", "Impseqrob", "QRILC", "MLE", "MinDet", "MinProb", "PI", "min", "halfmin", "zero", "nbavg", "with", "limpa", "none". "halfmin" is the DIA-NN half-minimum recipe: since the assay is on the log2 scale, halving the intensity means one unit below the global observed minimum.
<code>mar_method</code>	MAR method for combo/softHybrid mode (default: "Impseqrob")
<code>mnar_method</code>	MNAR method for combo/softHybrid mode (default: "min")
<code>prop_na_mnar</code>	NA proportion threshold for MNAR classification (default: 0.51)
<code>prop_present_mar</code>	Present proportion for MAR (default: 0.5)
<code>min_present_mar</code>	Minimum present values for MAR (default: 1)
<code>require_n_conditions</code>	Number of conditions required with presence (default: 1)
<code>max_na_prop</code>	Maximum NA proportion above which a protein is removed before imputation. NULL (the default) disables the filter, so every method receives the same proteins.
<code>method_args</code>	Named list of per-method argument lists (e.g., <code>list(knn = list(k = 10)</code> , <code>bpca = list(nPcs = 3)))</code>)

with_value Constant value for imp_method="with" (default: NA_real_)
 verbose Print progress messages (default: TRUE)

Details

- imp_method = "none": no imputation
- imp_method = "combo": two-stage MAR+MNAR with binary classification
- imp_method = "softHybrid": sigmoid-weighted MAR+MNAR blend (continuous)
- Any other method: apply to all NAs (no MAR/MNAR distinction)

Value

List with:

- se: SummarizedExperiment with imputed assay added
- x_imputed: Raw imputed matrix (for direct export)
- prefilter_summary: Pre-filtering summary
- imputation_summary: Imputation statistics
- mnar_mask: MNAR mask matrix (combo only, NULL otherwise)
- mar_mask: MAR mask matrix (combo only, NULL otherwise)

Examples

```
data(nadia_dia)
norm <- normalize_proteomics(nadia_dia, norm_method = "cycloess",
                             verbose = FALSE)

# Default: two-stage combo, Impseqrob for MAR and min for MNAR
imp <- impute_proteomics(norm$se, normalized_assay_name = "cycloess",
                         verbose = FALSE)
SummarizedExperiment::assayNames(imp$se)
sum(is.na(SummarizedExperiment::assay(imp$se, "Impseqrob_min")))

# Any single method works too, and per-method options go in method_args
imp2 <- impute_proteomics(norm$se, normalized_assay_name = "cycloess",
                           imp_method = "knn",
                           method_args = list(knn = list(k = 15)),
                           verbose = FALSE)
SummarizedExperiment::assayNames(imp2$se)
```

`merge_significance_info`*Merge significance information from DEPs_results*

Description

Converts DEPs_results (long format) into wide format with one adjP_* column per comparison, and computes sig_any.

Usage

```
merge_significance_info(  
  feature_ids,  
  DEPs_results,  
  assay_name,  
  alpha = 0.05,  
  verbose = TRUE  
)
```

Arguments

feature_ids	Vector of feature IDs
DEPs_results	DataFrame with columns Protein.IDs, adj.P.Val, Comparison, Assay
assay_name	Name of the assay used to filter DEPs_results
alpha	Significance threshold (default: 0.05)
verbose	Print progress messages (default: TRUE)

Value

DataFrame with FeatureID, adjP_*, sig_any

`nadia_add_pattern_profiler`*Add a Pattern Profiler run to an existing .nadia file*

Description

The clustering is an optional extra step, so it is written separately: this adds it to a file that already holds the analysis, and refuses to create one from scratch.

Usage

```
nadia_add_pattern_profiler(
  file,
  pattern_profiler,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

file	Path to an existing .nadia file.
pattern_profiler	Object from pattern_profiler_analysis() .
overwrite	Replace a clustering already in the file.
verbose	Print progress.

Value

The path, invisibly.

See Also

[write_nadia\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  f <- file.path(tempdir(), "pp.nadia")
  write_nadia(f, nadia_dia, res, verbose = FALSE)

  # pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results)
  # nadia_add_pattern_profiler(f, pp)

  unlink(f)
}
```

nadia_connect

Open a connection to a .nadia file

Description

For querying the file directly with SQL or dplyr. Most users want [read_nadia\(\)](#) instead; this is the way in when the file is larger than memory or when a specific query beats loading everything.

Usage

```
nadia_connect(file, read_only = TRUE)
```

Arguments

file	Path to a .nadia file.
read_only	Open without write access. TRUE by default, which is also what allows several processes to read the same file at once.

Details

The caller owns the connection and must close it with `DBI::dbDisconnect(con, shutdown = TRUE)`.

Value

A DBI connection.

See Also

[read_nadia\(\)](#), [nadia_tables\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  f <- file.path(tempdir(), "query.nadia")
  write_nadia(f, nadia_dia,
    process_proteomics(nadia_dia, verbose = FALSE),
    verbose = FALSE)

  con <- nadia_connect(f)
  DBI::dbGetQuery(con, "SELECT COUNT(*) AS n FROM proteins")
  DBI::dbDisconnect(con, shutdown = TRUE)

  unlink(f)
}
```

nadia_deps_with_clusters

Differential abundance with clusters, from a .nadia file

Description

The file's own answer to [deps_with_clusters\(\)](#). The join lives in the file as the view `v_deps_pattern_profiler`, so this reads a table rather than building one, and a client that is not R gets the same table from the same SQL. The filters are the ones [deps_with_clusters\(\)](#) applies, and the two functions return identical results for the same arguments.

Usage

```
nadia_deps_with_clusters(
  db,
  assignment = c("primary", "all"),
  min_membership = NULL,
  comparison = NULL,
  significant_only = FALSE
)
```

Arguments

db	A nadia_db from read_nadia() , or the path to a .nadia file.
assignment	"primary" (default) for one row per protein and comparison, carrying the cluster of highest membership; "all" for one row per protein, comparison and cluster.
min_membership	Optional extra membership threshold. It can only be stricter than the one the analysis ran with.
comparison	Optional character vector of comparisons to keep.
significant_only	Keep only the rows classified Up or Down.

Details

The table is read on demand rather than by [read_nadia\(\)](#), because it is as long as the number of proteins times comparisons times clusters and there is no reason to carry it around unasked.

Value

A data frame with the columns of DEPs_results followed by Cluster, Membership and ClusterRank, or NULL if the file holds no differential-abundance results. A file with results but no clustering returns the same shape with the three cluster columns NA.

See Also

[deps_with_clusters\(\)](#), [nadia_add_pattern_profiler\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  f <- file.path(tempdir(), "functional.nadia")
  write_nadia(f, nadia_dia, res, verbose = FALSE)

  head(nadia_deps_with_clusters(f))

  unlink(f)
}
```

nadia_dia

Example DIA experiment, already preprocessed

Description

Output of `preprocess_spectronaut()` on a trimmed Spectronaut report, ready to feed `process_proteomics()` and the rest of the pipeline without having to read the source file again.

Usage

```
data(nadia_dia)
```

Format

An S3 object of class `c("spectronaut_data", "proteomics_data", "list")` with three elements:

`metadata` Data frame of 12 rows and 7 columns, one row per injection: `R.FileName`, `R.Condition`, `R.Replicate`, `Coding` and the three identification counts reported by Spectronaut (`R.PrecursorsIdentified`, `R.StrippedSequencesIdentified`, `R.ProteinGroupsIdentified`).

`protein_id` Data frame of 2,000 rows by 52 columns holding the identification information of every protein group.

`protein_quant` Data frame of 2,000 rows by 44 columns: eight annotation columns plus, for each of the 12 samples, the number of precursors and of stripped sequences used for quantification and the intensity (`PG.Quantity_<condition>_<replicate>`).

Details

The data come from a quantitative proteomics experiment acquired on an Orbitrap Exploris at the Proteomics Facility of the Centro Nacional de Biotecnología (CNB-CSIC) and processed with Spectronaut v20. The original experiment has 16 injections of the same digest split across four conditions (A-D) of four replicates each, and 10,437 protein groups.

So that the package examples run quickly, three conditions (A, B and D) and a random sample of 2,000 protein groups are kept. Three and not two conditions are retained because the soft-clustering in `pattern_profiler_analysis()` groups profiles across conditions, and with only two points per profile there is no pattern left to cluster.

The sample is drawn at random, with a fixed seed, rather than picking the best-covered proteins. That matters: **8.1 % of the intensities are missing and 72.2 % of the proteins are complete**, closely matching the 7.9 % and 72.1 % of the full experiment. Selecting the most-quantified proteins instead would yield a dataset with no missing values at all, in which every imputation method returns the input unchanged, and in which 87 % of the proteins come out differentially expressed instead of the 31 % of the full data.

The full recipe is in `system.file("scripts", "make_extdata.R", package = "NADIA")`. The trimmed report this object is derived from is also distributed as a file, in `system.file("extdata", "nadia_dia_report.tsv.gz", package = "NADIA")`, so that `preprocess_spectronaut()` can be run end to end.

Source

Proteomics Facility, Centro Nacional de Biotecnología (CNB-CSIC).

Examples

```
data(nadia_dia)
nadia_dia

# The three conditions and their four replicates
table(nadia_dia$metadata$R.Condition)

# Proteins and intensity columns
dim(nadia_dia$protein_quant)
grep("^PG.Quantity_", colnames(nadia_dia$protein_quant), value = TRUE)

# Missing values, which is what the package is about. A Spectronaut report is
# long, one row per run and protein group, and where a protein was not
# quantified the row is simply absent; the pivot to wide format is what turns
# those absences into NA.
quant <- as.matrix(nadia_dia$protein_quant[
  grep("^PG.Quantity_", colnames(nadia_dia$protein_quant))])
round(100 * mean(is.na(quant)), 1)
```

nadia_export_parquet *Export the tables of a .nadia file to Parquet*

Description

A way out to an open, stable format. DuckDB reads its own files back a long way, but Parquet is the safer bet for something that has to be readable in ten years by software nobody has written yet, and it is what a repository will expect alongside a publication.

Usage

```
nadia_export_parquet(db, dir, views = TRUE, verbose = TRUE)
```

Arguments

db	A nadia_db from read_nadia() , or the path to a file.
dir	Directory to write to; created if it does not exist.
views	Export the reconstructed canonical tables (TRUE, the default) or the underlying storage tables (FALSE).
verbose	Print progress.

Value

The paths written, invisibly.

See Also

[read_nadia\(\)](#), [write_nadia\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE)) {
  data(nadia_dia)
  f <- file.path(tempdir(), "export.nadia")
  write_nadia(f, nadia_dia, verbose = FALSE)

  out <- file.path(tempdir(), "nadia_parquet_export")
  nadia_export_parquet(f, out, verbose = FALSE)
  list.files(out)

  unlink(c(f, out), recursive = TRUE)
}
```

nadia_pattern_profiler

Rebuild the Pattern Profiler table from a .nadia file

Description

Rebuild the Pattern Profiler table from a .nadia file

Usage

```
nadia_pattern_profiler(db)
```

Arguments

db A nadia_db from [read_nadia\(\)](#).

Value

The long_output data frame, ready for [cluster_profile_highchart_list\(\)](#), or NULL if the file holds no clustering.

See Also

[read_nadia\(\)](#), [nadia_add_pattern_profiler\(\)](#)

Examples

```

if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  f <- file.path(tempdir(), "nopp.nadia")
  write_nadia(f, nadia_dia, verbose = FALSE)

  is.null(nadia_pattern_profiler(read_nadia(f)))

  unlink(f)
}

```

nadia_preprocessing	<i>Rebuild the preprocessing object from a .nadia file</i>
---------------------	--

Description

Returns an object indistinguishable from what the preprocess_*() function that wrote the file produced – Spectronaut, DIA-NN, TMT or label-free – so it can be fed straight to [process_proteomics\(\)](#) or to the table functions.

Usage

```
nadia_preprocessing(db)
```

Arguments

db A nadia_db from [read_nadia\(\)](#).

Value

A proteomics_data object with metadata, protein_id and protein_quant.

See Also

[read_nadia\(\)](#), [nadia_result\(\)](#)

Examples

```

if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  f <- file.path(tempdir(), "pre.nadia")
  write_nadia(f, nadia_dia, verbose = FALSE)

  pre <- nadia_preprocessing(read_nadia(f))
  identical(pre$protein_id, nadia_dia$protein_id)

  unlink(f)
}

```

nadia_result*Rebuild the processing result from a .nadia.file*

Description

Returns a `proteomics_result` with everything `process_proteomics()` returns, the `SummarizedExperiment` included, so that every plotting function in the package works from the file directly:

Usage

```
nadia_result(db)
```

Arguments

`db` A `nadia_db` from `read_nadia()`.

Details

```
db <- read_nadia("experiment.nadia")
res <- nadia_result(db)
volcano_highchart_list(res$DEPs_results)
pca_highchart_list(res$PCA_Input)
```

Value

A `proteomics_result` object, or an error if the file holds only the preprocessing step.

See Also

`read_nadia()`, `nadia_preprocessing()`

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  f <- file.path(tempdir(), "result.nadia")
  write_nadia(f, nadia_dia, res, verbose = FALSE)

  back <- nadia_result(read_nadia(f))
  identical(back$DEPs_results, res$DEPs_results)
  class(volcano_highchart_list(back$DEPs_results)[[1]])

  unlink(f)
}
```

nadia_skill_path	<i>Locate the NADIA agent skill</i>
------------------	-------------------------------------

Description

Returns the directory holding the agent skill that ships with NADIA. The skill documents the analysis pipeline for a coding assistant: which function to call at each stage, how to choose between the normalisation and imputation methods, what to verify afterwards, and the mistakes the package's defaults invite. It is plain Markdown, so any assistant can read it, and it follows the layout Claude expects for a skill: a SKILL.md entry point that points at the files under reference/ and scripts/.

Usage

```
nadia_skill_path(...)
```

Arguments

... Character path segments below the skill directory, passed on to `base::system.file()`. With no arguments the directory itself is returned.

Details

To use it with Claude Code, copy the directory into the skills folder of a project (`.claude/skills/nadia/`) or of the user (`~/.claude/skills/nadia/`). The examples below print the command that does it.

Value

A single character string: the absolute path, or `""` when the skill cannot be found, which is what `system.file()` returns for a missing file.

See Also

`vignette("NADIA")` for the same pipeline written for a person.

Examples

```
# Where the skill lives
nadia_skill_path()

# Its entry point, and the files it refers to
nadia_skill_path("SKILL.md")
list.files(nadia_skill_path(), recursive = TRUE)

# Install it for Claude Code, for the current user
cat(sprintf(
  "cp -R '%s' ~/.claude/skills/nadia",
  nadia_skill_path()
))
```

nadia_tables	<i>List the contents of a .nadia file without loading it</i>
--------------	--

Description

List the contents of a .nadia file without loading it

Usage

```
nadia_tables(file)
```

Arguments

file	Path to a .nadia file.
------	------------------------

Value

A data frame with name, type (table or view) and rows.

See Also

[read_nadia\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {  
  data(nadia_dia)  
  f <- file.path(tempdir(), "tables.nadia")  
  write_nadia(f, nadia_dia, verbose = FALSE)  
  
  nadia_tables(f)  
  
  unlink(f)  
}
```

nm_compute_metrics	<i>Compute quantitative group-separation metrics per normalization method</i>
--------------------	---

Description

Iterates over assays in a SummarizedExperiment and computes ten metrics that quantify how well the normalization separates sample groups.

Usage

```
nm_compute_metrics(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  seed = 42L,
  ...
)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>seed</code>	Integer. Seed for the two metrics that use randomness: the Hopkins statistic, which samples random reference points, and the PERMANOVA p-value, which permutes the group labels. Without it neither column is reproducible between runs. The caller's RNG state is restored on exit. Default 42.
<code>...</code>	Additional arguments (unused).

Details

- **PC1_VarPct**: % variance explained by PC1 (higher = more structure).
- **PC1_F_ratio**: ANOVA F-ratio on PC1 scores (higher = better separation).
- **PERMANOVA_R2**: Proportion of variance explained by grouping (requires `vegan`; NA if not installed).
- **Silhouette_mean**: Mean silhouette width (requires `cluster`; NA if not installed).
- **MDS_GOF**: Goodness-of-fit of 2D MDS projection. Note: in benchmarking context, *lower* values tend to indicate better normalization (a good method preserves multi-dimensional biological variability that 2D cannot capture).
- **MDS_CophCor**: Cophenetic correlation between original and MDS distances. Same caveat as **MDS_GOF**: *lower* values tend to correlate with better benchmarking performance.
- **Spectral_Entropy**: Normalized Shannon entropy of PCA eigenvalues (0-1). Values near 0 = variance concentrated in few PCs; near 1 = uniform spread.
- **CumVar_PC2**: Cumulative % variance explained by PC1 + PC2 (0-100).
- **Hopkins**: Hopkins statistic for clustering tendency (0-1). Values > 0.5 suggest non-random cluster structure.
- **Condition_Number**: Ratio of largest to smallest PCA eigenvalue (≥ 1). High values indicate multicollinearity or numerical instability.

Value

A data.frame with one row per method and columns: `Method`, `PC1_VarPct`, `PC1_F_ratio`, `PERMANOVA_R2`, `PERMANOVA_pval`, `Silhouette_mean`, `MDS_GOF`, `MDS_CophCor`, `Spectral_Entropy`, `CumVar_PC2`, `Hopkins`, `Condition_Number`.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

metrics_df <- nm_compute_metrics(se)
metrics_df[, c("Method", "PC1_VarPct", "PC1_F_ratio", "Silhouette_mean")]
```

nm_plot_boxplot *Intensity boxplot per method*

Description

Displays distribution of protein intensities for each sample, faceted by normalization method.

Usage

```
nm_plot_boxplot(se, assay_names = NULL, condition_col = "Condition", ...)
```

Arguments

se	SummarizedExperiment from <code>import_norm_matrices()</code> .
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in <code>colData</code> with condition labels.
...	Additional arguments (unused).

Value

ggplot object.

Examples

[illegible]

nm_plot_correlation *Intra-group correlation boxplot per method (PRONE-style)*

Description

Pairwise within-group correlations for each method shown as boxplots with error bars. Higher and less variable values indicate better normalization.

Usage

```
nm_plot_correlation(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  cor_method = "pearson",
  ...
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
cor_method	Correlation method passed to cor(): "pearson", "spearman", or "kendall". Default "pearson".
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_correlation(se, cor_method = "pearson")
```

nm_plot_density	<i>Density plot per method</i>
-----------------	--------------------------------

Description

KDE curves for each sample's intensity distribution, faceted by method.

Usage

```
nm_plot_density(se, assay_names = NULL, condition_col = "Condition", ...)
```

Arguments

se	SummarizedExperiment from <code>import_norm_matrices()</code> .
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_density(se)
```

nm_plot_final_ranking	<i>Horizontal bar chart of combined final ranking</i>
-----------------------	---

Description

Produces a horizontal bar chart with normalization methods ordered by ascending final rank (best at top). Uses the same PRONE-style palette as other `nm_plot_*()` functions.

Usage

```
nm_plot_final_ranking(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  cor_method = "pearson",  
  ...  
)
```

Arguments

- se SummarizedExperiment from import_norm_matrices().
- assay_names Character vector of assay names to include. NULL = all.
- condition_col Column in colData with condition labels.
- cor_method Correlation method forwarded to nm_rank_final().
- ... Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)  
se <- nm_prepare_se(nadia_dia, verbose = FALSE)  
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),  
                           verbose = FALSE)  
  
nm_plot_final_ranking(se)
```

nm_plot_mds	<i>MDS 2D scatter per method</i>
-------------	----------------------------------

Description

Classical MDS on Euclidean distances of scaled, complete-case data. Faceted by method, colored by condition.

Usage

```
nm_plot_mds(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  mds_scales = c("free", "fixed"),  
  ...  
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
mds_scales	Facet scaling: "free" (default) allows independent axes per panel; "fixed" forces shared axes for easier cross-method comparison.
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_mds(se)
```

nm_plot_mds1_ranking *Horizontal bar chart of MDS1 variance ranking*

Description

Produces a horizontal bar chart with normalization methods ordered by ascending MDS1 variance percentage (lower = better normalization). Uses the same PRONE-style palette as other nm_plot_*() functions.

Usage

```
nm_plot_mds1_ranking(se, assay_names = NULL, condition_col = "Condition", ...)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_mds1_ranking(se)
```

nm_plot_metrics

*Bar chart of group-separation metrics per normalization method***Description**

Calls `nm_compute_metrics()` internally and produces a faceted bar chart (one facet per metric, free y-scales) with labeled values.

Usage

```
nm_plot_metrics(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  seed = 42L,
  ...
)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>seed</code>	Integer. Seed for the two metrics that use randomness: the Hopkins statistic, which samples random reference points, and the PERMANOVA p-value, which permutes the group labels. Without it neither column is reproducible between runs. The caller's RNG state is restored on exit. Default 42.
<code>...</code>	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_metrics(se)
```

nm_plot_pc1_ranking	<i>Horizontal bar chart of PC1 variance ranking</i>
---------------------	---

Description

Produces a horizontal bar chart with normalization methods ordered by descending PC1 variance percentage. Uses the same PRONE-style palette as other `nm_plot_*`() functions.

Usage

```
nm_plot_pc1_ranking(se, assay_names = NULL, condition_col = "Condition", ...)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>...</code>	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_pc1_ranking(se)
```

nm_plot_pca

*PCA scatter plot per method***Description**

PC1 vs PC2 scatter, colored by condition, faceted by method. Percentage of variance explained shown on each axis.

Usage

```
nm_plot_pca(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  pca_scales = c("free", "fixed"),
  ...
)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. NULL = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>pca_scales</code>	Facet scaling: "free" (default) allows independent axes per method; "fixed" uses shared axes to compare separation magnitude.
<code>...</code>	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_pca(se, pca_scales = "fixed")
```

nm_plot_pcv

*PCV boxplot per method (PRONE-style)***Description**

One boxplot per normalization method showing the per-protein CV distribution (CV averaged across groups for each protein). Lower and less spread values indicate better within-group consistency. With `diff = TRUE`, shows % reduction vs baseline as a bar chart.

Usage

```
nm_plot_pcv(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  diff = FALSE,
  baseline = "log2",
  ...
)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>diff</code>	Logical. If <code>TRUE</code> , show % reduction vs baseline. Default <code>FALSE</code> .
<code>baseline</code>	Character. Assay name used as reference for diff mode. Default <code>"log2"</code> .
<code>...</code>	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
  verbose = FALSE)

nm_plot_pcv(se)

# Percent CV reduction relative to the unnormalized log2 baseline
nm_plot_pcv(se, diff = TRUE, baseline = "log2")
```


nm_plot_pmad *PMAD boxplot per method (PRONE-style)*

Description

One boxplot per normalization method showing the per-protein MAD distribution (MAD averaged across groups for each protein). Lower values indicate better within-group consistency. With diff = TRUE, shows % reduction vs baseline as a bar chart.

Usage

```
nm_plot_pmad(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  diff = FALSE,
  baseline = "log2",
  ...
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
diff	Logical. If TRUE, show % reduction vs baseline. Default FALSE.
baseline	Character. Assay name used as reference for diff mode. Default "log2".
...	Additional arguments (unused).

Value

ggplot object.

Examples

[illegible]

```
nm_plot_pmad(se)
```

nm_plot_qq	<i>Q-Q plot per method (NormalyzerDE style)</i>
------------	---

Description

Quantile-quantile plot against a normal distribution for a single sample, one facet per normalization method. Default uses the first column (as NormalyzerDE does).

Usage

```
nm_plot_qq(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  which_sample = NULL,
  ...
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
which_sample	Character. Name of the sample to inspect. Default = first column of se.
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_qq(se, which_sample = "A_1")
```

nm_plot_scatter	<i>Sample-vs-sample scatter plot per method (NormalyzerDE style)</i>
-----------------	--

Description

Plots log2-intensity of one sample against another, one point per protein, for each normalization method. A linear fit and the adjusted R^2 are overlaid. Default comparison uses the first two columns of the SE (as NormalyzerDE does).

Usage

```
nm_plot_scatter(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  sample1 = NULL,
  sample2 = NULL,
  ...
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
sample1	Character. Name of the first sample (x-axis). Default = first column of se.
sample2	Character. Name of the second sample (y-axis). Default = second column of se.
...	Additional arguments (unused).

Value

ggplot object.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_plot_scatter(se, sample1 = "A_1", sample2 = "A_2")
```

nm_prepare_se

*Prepare a SummarizedExperiment from a proteomics_data object***Description**

Convenience wrapper that extracts metadata and protein data from a `proteomics_data` object (from any of the `preprocess_*`() functions), performs zero-to-NA conversion, protein filtering, and returns a SE with assays "raw" and "log2" – ready for `nm_run_normalizations()` or `normalization_metrics(..., methods = "all")`.

Usage

```
nm_prepare_se(
  preprocessing,
  min_reps = NULL,
  min_groups = 1,
  covariate_df = NULL,
  verbose = TRUE
)
```

Arguments

<code>preprocessing</code>	A <code>proteomics_data</code> object, from any of the <code>preprocess_*</code> () functions.
<code>min_reps</code>	Minimum replicates with non-NA values per group for protein filtering. If NULL, auto-computed as half the smallest group. Default NULL.
<code>min_groups</code>	Minimum groups meeting <code>min_reps</code> (default: 1).
<code>covariate_df</code>	Optional covariate data.frame for paired designs (must contain a <code>Column</code> column). Default NULL.
<code>verbose</code>	Logical. Print progress messages. Default TRUE.

Details

Internally calls `normalize_proteomics()` with `norm_method = "log2"` (no additional normalization).

Value

SummarizedExperiment with assays "raw" and "log2".

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
dim(se)
SummarizedExperiment::assayNames(se)
```

nm_rank_cor	<i>Rank normalization methods by median intragroup correlation (descending – higher is better)</i>
-------------	--

Description

Rank normalization methods by median intragroup correlation (descending – higher is better)

Usage

```
nm_rank_cor(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  cor_method = "pearson",  
  verbose = TRUE  
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
cor_method	Correlation method: "pearson", "spearman", or "kendall".
verbose	Logical. Print progress messages. Default TRUE.

Value

A data.frame with columns Method, Median_Cor, Rank.

Examples

```
data(nadia_dia)  
se <- nm_prepare_se(nadia_dia, verbose = FALSE)  
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),  
                           verbose = FALSE)  
  
nm_rank_cor(se, cor_method = "pearson", verbose = FALSE)
```

nm_rank_final	<i>Compute a combined final ranking across PCV, PMAD, PEV, Correlation and group-separation (PC1 F-ratio)</i>
---------------	---

Description

For each method, the final rank is the mean of five individual ranks: four intragroup-precision metrics (pcv, pmad, pev, cor) plus a group-separation metric (Rank_Sep, the ANOVA F-ratio on PC1 scores, higher = better separation). Lower final rank = better overall normalization quality.

Usage

```
nm_rank_final(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  cor_method = "pearson",
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment from <code>import_norm_matrices()</code> .
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
cor_method	Correlation method forwarded to <code>nm_rank_cor()</code> .
verbose	Logical. Print progress messages. Default TRUE.

Details

Note: earlier versions averaged Rank_PC1 (PC1 variance %, higher = better) and Rank_MDS1 (MDS1 variance %, lower = better). Both capture the same dominant axis of variation but with opposite directions, so they partially cancelled within the mean. They are replaced here by a single, unambiguous group-separation rank based on the PC1 F-ratio. The standalone `nm_rank_pc1()` / `nm_rank_mds1()` functions are unchanged for individual inspection.

Value

A data.frame with columns: Method, Rank_PCV, Rank_PMad, Rank_PEV, Rank_Cor, Rank_Sep, Rank_Final, ordered by Rank_Final ascending (best first).

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)
```

```
nm_rank_final(se, verbose = FALSE)
```

nm_rank_mds1

Rank normalization methods by MDS1 variance explained

Description

For each assay in `se`, computes the percentage of variance captured by the first MDS dimension (using only positive eigenvalues from classical MDS) and returns a ranking ordered by ascending MDS1 variance (lower = better).

Usage

```
nm_rank_mds1(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  verbose = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> = all.
<code>condition_col</code>	Column in <code>colData</code> with condition labels.
<code>verbose</code>	Logical. Print progress messages. Default <code>TRUE</code> .

Value

A `data.frame` with columns `Method`, `MDS1_VarPct`, `Rank`, ordered by `MDS1_VarPct` ascending (lower = better; rank 1 = best).

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_rank_mds1(se, verbose = FALSE)
```

nm_rank_pc1*Rank normalization methods by PC1 variance explained*

Description

Computes the percentage of total variance captured by PC1 for each assay in a SummarizedExperiment and returns a data.frame sorted in descending order. Higher PC1 variance generally indicates stronger group separation.

Usage

```
nm_rank_pc1(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  verbose = TRUE  
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
verbose	Logical. Print progress messages. Default TRUE.

Value

A data.frame with columns Method, PC1_VarPct, Rank, ordered by PC1_VarPct descending.

Examples

```
data(nadia_dia)  
se <- nm_prepare_se(nadia_dia, verbose = FALSE)  
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),  
                           verbose = FALSE)  
  
nm_rank_pc1(se, verbose = FALSE)
```

nm_rank_pcv	<i>Rank normalization methods by median PCV (ascending – lower is better)</i>
-------------	---

Description

Rank normalization methods by median PCV (ascending – lower is better)

Usage

```
nm_rank_pcv(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  verbose = TRUE  
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
verbose	Logical. Print progress messages. Default TRUE.

Value

A data.frame with columns Method, Median_PCV, Rank.

Examples

```
data(nadia_dia)  
se <- nm_prepare_se(nadia_dia, verbose = FALSE)  
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),  
                           verbose = FALSE)  
  
nm_rank_pcv(se, verbose = FALSE)
```

nm_rank_pev	<i>Rank normalization methods by median PEV (ascending – lower is better)</i>
-------------	---

Description

Rank normalization methods by median PEV (ascending – lower is better)

Usage

```
nm_rank_pev(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
verbose	Logical. Print progress messages. Default TRUE.

Value

A data.frame with columns Method, Median_PEV, Rank.

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                           verbose = FALSE)

nm_rank_pev(se, verbose = FALSE)
```

nm_rank_pmad	<i>Rank normalization methods by median PMAD (ascending – lower is better)</i>
--------------	--

Description

Rank normalization methods by median PMAD (ascending – lower is better)

Usage

```
nm_rank_pmad(  
  se,  
  assay_names = NULL,  
  condition_col = "Condition",  
  verbose = TRUE  
)
```

Arguments

se	SummarizedExperiment from import_norm_matrices().
assay_names	Character vector of assay names to include. NULL = all.
condition_col	Column in colData with condition labels.
verbose	Logical. Print progress messages. Default TRUE.

Value

A data.frame with columns Method, Median_PMad, Rank.

Examples

```
data(nadia_dia)  
se <- nm_prepare_se(nadia_dia, verbose = FALSE)  
se <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),  
                           verbose = FALSE)  
  
nm_rank_pmad(se, verbose = FALSE)
```

nm_run_normalizations *Run multiple normalization methods from a baseline assay*

Description

Takes a SummarizedExperiment with a log2-scale assay and applies each requested normalization method, returning a new SE with one assay per method.

Usage

```
nm_run_normalizations(
  se,
  assay_name = "log2",
  methods = "all",
  method_args = list(),
  include_baseline = TRUE,
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment with at least one log2-scale assay.
assay_name	Name of the baseline assay to normalize from. Default "log2".
methods	Character vector of method names, or "all" for all 12 benchmark methods. Default "all".
method_args	Named list of per-method arguments. E.g. list(cycloess = list(method = "fast", span = 0.8)).
include_baseline	Logical. Include the baseline assay in the output SE. Default TRUE.
verbose	Logical. Print progress messages. Default TRUE.

Value

SummarizedExperiment with one assay per successfully normalized method (plus baseline if include_baseline = TRUE).

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)
se_bench <- nm_run_normalizations(se, methods = c("cycloess", "quantile", "MAD"),
                                verbose = FALSE)
SummarizedExperiment::assayNames(se_bench)
```

normalization_metrics *Generate quality metric plots for normalization comparison*

Description

Calls all (or selected) `nm_plot_*()` functions on a `SummarizedExperiment` produced by `import_norm_matrices()`. Each plot is wrapped in `tryCatch()` so that a failure in one plot does not abort the entire run.

Usage

```
normalization_metrics(
  se,
  assay_names = NULL,
  condition_col = "Condition",
  plots = "all",
  cor_method = "pearson",
  pca_scales = c("free", "fixed", "both"),
  mds_scales = c("free", "fixed", "both"),
  methods = NULL,
  method_args = list(),
  base_assay = "log2",
  seed = 42L,
  verbose = TRUE,
  output_dir = NULL,
  export_plots = TRUE,
  export_tables = TRUE,
  plot_width = 12,
  plot_height = 8,
  plot_dpi = 150
)
```

Arguments

<code>se</code>	<code>SummarizedExperiment</code> from <code>import_norm_matrices()</code> .
<code>assay_names</code>	Character vector of assay names to include. <code>NULL</code> (default) = all assays.
<code>condition_col</code>	Column name in <code>colData(se)</code> with condition labels. Default <code>"Condition"</code> .
<code>plots</code>	Character vector of plot names to generate, or <code>"all"</code> (default). Valid names: <code>"boxplot"</code> , <code>"density"</code> , <code>"pcv"</code> , <code>"pmad"</code> , <code>"pev"</code> , <code>"pca"</code> , <code>"correlation"</code> , <code>"mds"</code> , <code>"scatter"</code> , <code>"qq"</code> , <code>"metrics"</code> , <code>"pc1_ranking"</code> , <code>"mds1_ranking"</code> , <code>"final_ranking"</code> . When <code>pca_scales = "both"</code> , <code>"pca"</code> expands to <code>"pca_free" + "pca_fixed"</code> . When <code>mds_scales = "both"</code> , <code>"mds"</code> expands to <code>"mds_free" + "mds_fixed"</code> .
<code>cor_method</code>	Correlation method for <code>nm_plot_correlation()</code> . Default <code>"pearson"</code> .
<code>pca_scales</code>	Facet scaling for PCA plot: <code>"free"</code> (default), <code>"fixed"</code> , or <code>"both"</code> to generate and export both variants (<code>pca_free</code> and <code>pca_fixed</code> in the returned list).
<code>mds_scales</code>	Facet scaling for MDS plot: <code>"free"</code> (default), <code>"fixed"</code> , or <code>"both"</code> to generate and export both variants (<code>mds_free</code> and <code>mds_fixed</code> in the returned list).

methods	Character vector of normalization method names to auto-benchmark, "all" for the 12 benchmark methods ("log2" is the baseline and is excluded, so this is one fewer than the 13 that <code>normalize_proteomics()</code> accepts), or NULL (default) to skip auto-normalization and use existing assays.
method_args	Named list of per-method arguments forwarded to <code>nm_run_normalizations()</code> . E.g. <code>list(cycloess = list(method = "fast", span = 0.8))</code> .
base_assay	Name of the baseline assay to normalize from when using auto-normalization. Default "log2".
seed	Integer. Seed passed to <code>nm_compute_metrics()</code> for the two metrics that use randomness, the Hopkins statistic and the PERMANOVA p-value. Default 42.
verbose	Logical. Print progress messages. Default TRUE.
output_dir	Character. Path to export directory. If NULL (default), no files are exported. When set, tables (TSV) and/or plots (PNG) are saved to this directory (created if needed).
export_plots	Logical. Export plots as PNG when <code>output_dir</code> is set. Default TRUE.
export_tables	Logical. Export tables as TSV when <code>output_dir</code> is set. Default TRUE.
plot_width	Numeric. Width in inches for exported plots. Default 12.
plot_height	Numeric. Height in inches for exported plots. Default 8.
plot_dpi	Numeric. Resolution for exported plots. Default 150.

Details

When `methods` is provided, the function first runs `nm_run_normalizations()` to automatically apply the requested normalization methods from the `base_assay`, then computes metrics and generates plots on the resulting multi-assay SE.

Value

Named list of ggplot objects (or NULL for failed plots), plus data.frames always computed regardless of plots selection: `metrics_table` (from `nm_compute_metrics()`), `pc1_rank`, `mds1_rank`, `pcv_rank`, `pmad_rank`, `pev_rank`, `cor_rank`, and `final_rank` (combined ranking as the mean of the five ranks `Rank_PCV`, `Rank_PMA`, `Rank_PEV`, `Rank_Cor` and `Rank_Sep`; `pc1_rank` and `mds1_rank` are standalone diagnostics and do not enter it).

Examples

```
data(nadia_dia)
se <- nm_prepare_se(nadia_dia, verbose = FALSE)

# Auto-benchmark: normalize from the "log2" baseline, then score the methods
res <- normalization_metrics(se, methods = c("cycloess", "quantile", "MAD"),
                             plots = c("pcv", "final_ranking"), verbose = FALSE)

res$final_rank
res$pcv
```

normalize_proteomics *Normalize proteomics data*

Description

Complete normalization pipeline: zero-to-NA conversion, protein filtering by group presence, SummarizedExperiment creation, and normalization with the selected method. Output is always in log2 scale.

Usage

```
normalize_proteomics(
  data,
  metadata = NULL,
  min_reps = NULL,
  min_groups = 1,
  norm_method = "cycloess",
  cyclic_loess_method = c("fast", "pairs"),
  cyclic_loess_iterations = 3,
  cyclic_loess_span = 0.7,
  verbose = TRUE
)
```

Arguments

data	Either a proteomics_data object – the output of <code>preprocess_spectronaut()</code> , <code>preprocess_diann()</code> , <code>preprocess_tmt()</code> or <code>preprocess_lfq()</code> – in which case metadata is derived from it and must be left NULL, or a data frame with ProteinGroups, GeneNames, UniqPepts and the intensity columns.
metadata	Data frame with Column, Condition, Replicate. Only needed when data is a data frame.
min_reps	Minimum replicates for filtering. NULL = auto: $\text{floor}(\text{min_group_size} / 2)$
min_groups	Minimum groups meeting min_reps (default: 1)
norm_method	Normalization method (default: "cycloess"). One of: • Group A (input: raw intensities): "log2Norm", "GlobalMedian", "GlobalMean", "eqmedians", "vsn" • Group B (input: log2 assay): "log2" (no extra normalization), "quantile", "Rlr", "MAD", "cycloess", "medianNorm", "meanNorm", "quantile.robust"
cyclic_loess_method	Cyclic Loess method: "fast" or "pairs" (default: "fast")
cyclic_loess_iterations	Number of iterations for Cyclic Loess (default: 3)
cyclic_loess_span	Span parameter for Cyclic Loess (default: 0.7)
verbose	Print progress messages (default: TRUE)

Value

List with:

- se: SummarizedExperiment with assays raw, log2, and <norm_method> (assays raw + log2 only when norm_method = "log2")
- filter_summary: Filtering summary
- na_overview: NA statistics

Examples

```
data(nadia_dia)

# The object returned by preprocess_*() is accepted directly
norm <- normalize_proteomics(nadia_dia, norm_method = "cycloess",
                             verbose = FALSE)
SummarizedExperiment::assayNames(norm$se)

# Zeros become NA and low-coverage proteins are filtered out
norm$filter_summary

# Any of the 13 methods can be used
norm2 <- normalize_proteomics(nadia_dia, norm_method = "quantile",
                              verbose = FALSE)
SummarizedExperiment::assayNames(norm2$se)
```

pattern_profiler_analysis

Pattern Profiler Analysis

Description

Complete clustering pipeline starting from a SummarizedExperiment. Writes a parquet file in LONG format for visualisation.

Usage

```
pattern_profiler_analysis(
  se_proc,
  DEPs_results,
  assay_name = NULL,
  filter_mode = c("any", "all", "specific"),
  alpha = 0.05,
  comparison = NULL,
  condition_order = NULL,
  aggregate = c("median", "mean"),
  c_range = 2:10,
```

```

    auto_select_c = TRUE,
    c = NULL,
    selection_method = c("xb", "consensus", "elbow"),
    min_membership = 0.25,
    seed = 42,
    seeds = c(42, 123, 456),
    output_file = NULL,
    verbose = TRUE
)

```

Arguments

<code>se_proc</code>	SummarizedExperiment with intensity data
<code>DEPs_results</code>	DataFrame with differential expression results (must have an 'Assay' column)
<code>assay_name</code>	Name of the assay to cluster. It is also the value used to filter <code>DEPs_results</code> by its Assay column, so the two must agree. With the default NULL it is taken from <code>DEPs_results\$Assay</code> , which records the assay the differential abundance was computed on; supply it explicitly when those results cover more than one assay.
<code>filter_mode</code>	Filtering mode: "any" (significant in at least one comparison), "all" (ALL features, with no significance filtering), "specific" (significant in the comparison given by comparison)
<code>alpha</code>	Significance threshold (default: 0.05)
<code>comparison</code>	Specific comparison (for <code>filter_mode="specific"</code>)
<code>condition_order</code>	Condition order for the profiles
<code>aggregate</code>	Aggregation method: "median" or "mean"
<code>c_range</code>	Range of cluster numbers to evaluate
<code>auto_select_c</code>	Automatic selection of the number of clusters (default: TRUE)
<code>c</code>	Fixed number of clusters (if <code>auto_select_c=FALSE</code>)
<code>selection_method</code>	Selection method: "xb", "consensus", "elbow"
<code>min_membership</code>	Minimum membership threshold for inclusion in the output
<code>seed</code>	Integer. Seed for the Mfuzz clustering itself. Fuzzy c-means starts from a random partition, so the cluster labels and the memberships depend on it; fix it to make a run reproducible. The caller's RNG state is restored on exit. Default 42.
<code>seeds</code>	Numeric vector of seeds used when <code>auto_select_c = TRUE</code> : each candidate number of clusters is fitted once per seed and the metrics are averaged, so that the choice of <code>c</code> does not hang on a single partition. Default <code>c(42, 123, 456)</code> .
<code>output_file</code>	Path of the output parquet file. NULL by default, which writes nothing to disk; the long-format data is returned regardless in the <code>long_output</code> element of the result.
<code>verbose</code>	Show progress messages

Value

List (invisible) with the clustering results: optimal_c, m, conditions, feature counts, selection_metrics, cluster_counts, the Mfuzz cl object, the standardised eset_std and long_output, the long-format data.frame that is written when output_file is given.

Examples

```
if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  # auto_select_c = TRUE would cluster once per value of c_range
  pp <- pattern_profiler_analysis(
    res$se_proc,
    DEPs_results = res$DEPs_results,
    assay_name   = "Impseqrob_min",
    filter_mode  = "any",
    condition_order = c("A", "B", "D"),
    auto_select_c = FALSE,
    c             = 3,
    verbose       = FALSE
  )
  print(pp$cluster_counts)
  print(head(pp$long_output))
}
```

pca_covariates_plot *PCA Plots Colored by Covariates*

Description

Generates PCA scatter plots (PC1 vs PC2) from a SummarizedExperiment assay, coloring points by different covariates. Useful for visually identifying batch effects or confounding.

Usage

```
pca_covariates_plot(
  se,
  assay_name = NULL,
  covariates,
  na_action = "complete",
  fill_value = -1,
  center = TRUE,
  scale. = TRUE,
  de_results = NULL,
  comparison = "any",
```

```

    alpha = 0.05,
    filter_samples = FALSE,
    condition_column = "Condition",
    point_size = 3,
    verbose = TRUE
  )

```

Arguments

<code>se</code>	SummarizedExperiment object.
<code>assay_name</code>	Character. Assay to analyze. NULL = second assay (normalized, pre-imputation).
<code>covariates</code>	Character vector. Column names in <code>colData(se)</code> to use for coloring (e.g., <code>c("Injection", "Digestion", "Condition")</code>).
<code>na_action</code>	Character: "complete" (default) or "fill".
<code>fill_value</code>	Numeric. NA replacement when <code>na_action="fill"</code> (default -1).
<code>center</code>	Logical. Center before PCA (default TRUE).
<code>scale.</code>	Logical. Scale before PCA (default TRUE).
<code>de_results</code>	Data.frame with differential expression results (e.g., <code>result\$DEPs_results</code>). Must contain columns "Protein.IDs", "adj.P.Val", and "Comparison". When provided, PCA is computed using only significant proteins. Default NULL (all proteins).
<code>comparison</code>	Character. Which comparison to filter by. Use "any" (default) to keep proteins significant in any comparison, or specify a comparison name (e.g., "Post_vs_Pre") to filter to that one only. Ignored if <code>de_results</code> is NULL.
<code>alpha</code>	Numeric. Significance threshold for adj.P.Val when filtering by <code>de_results</code> (default 0.05). Ignored if <code>de_results</code> is NULL.
<code>filter_samples</code>	Logical. If TRUE and a specific comparison is given (not "any"), subset samples to only those belonging to the comparison conditions. Mirrors <code>pca_highchart_list(filter_samples_to_comp)</code> . Default FALSE (use all samples).
<code>condition_column</code>	Character. Column name in <code>colData(se)</code> containing condition labels, used when <code>filter_samples=TRUE</code> . Default "Condition".
<code>point_size</code>	Numeric. Size of scatter points (default 3).
<code>verbose</code>	Logical (default TRUE).

Details

Categorical covariates use a discrete color scale; numeric covariates use a continuous viridis gradient.

Value

Named list:

grid ggplot2 facet_wrap with one panel per covariate

plots Named list of individual ggplot2 objects per covariate

pca_summary data.frame with PC1/PC2 variance explained

assay_name Assay used

n_proteins Number of proteins used

Examples

```
data(nadia_dia)

# nadia_dia has no batch information, so the example builds a plausible one:
# two digestion batches crossed with the three conditions.
cov <- data.frame(Column = nadia_dia$metadata$Coding,
                  Batch = rep(c("b1", "b2"),
                             length.out = nrow(nadia_dia$metadata)),
                  stringsAsFactors = FALSE)
res <- process_proteomics(nadia_dia, covariate_df = cov, verbose = FALSE)

# How the samples lay out in PCA space against each covariate
p <- pca_covariates_plot(res$se_proc, assay_name = "Impseqrob_min",
                        covariates = c("Condition", "Batch"))
names(p)
```

pca_highchart

Interactive PCA Plot with Highcharts

Description

Builds a PCA scatter plot with the option of showing per-group ellipses or convex hulls, similar to factoextra::fviz_pca_ind.

Usage

```
pca_highchart(
  scores_df,
  color_by = "Condition",
  group_order = NULL,
  palette = NULL,
  title = NULL,
  addEllipses = TRUE,
  ellipse_type = c("convex", "confidence"),
  ellipse_level = 0.95,
  ellipse_fill_opacity = 0.12,
  ellipse_line_width = 1,
  ellipse_npoints = 100,
  point_size = 5,
  show_labels = FALSE,
  label_size = 10
)
```

Arguments

scores_df	Data frame produced by build_pca_scores() with columns: PC1, PC2, PC1_Perc, PC2_Perc, Subset, Condition, Replicate, SampleID
color_by	Column used to colour the points (default: "Condition")
group_order	Vector with the order of the groups/conditions (optional)
palette	Named vector of colours, or NULL for an automatic palette
title	Chart title (optional, defaults to Subset)
addEllipses	Show ellipses/hulls around the groups (default: TRUE)
ellipse_type	Ellipse type: "convex" for a convex hull or "confidence" for a confidence ellipse based on the normal distribution (default: "convex")
ellipse_level	Confidence level for ellipse_type = "confidence" (default: 0.95). Typical values: 0.95, 0.90, 0.68
ellipse_fill_opacity	Fill opacity (0-1, default: 0.12)
ellipse_line_width	Outline line width (default: 1)
ellipse_npoints	Number of points used to draw the confidence ellipse (default: 100). Only applies to ellipse_type = "confidence"
point_size	Point radius (default: 5)
show_labels	Show the point labels (SampleID) (default: FALSE)
label_size	Label font size in px (default: 10)

Value

A highchart object

Examples

```
# The scores are normally built internally by pca_highchart_list();
# this is the layout the function expects (one row per sample).
scores <- data.frame(
  SampleID = paste(rep(c("A", "B", "D"), each = 4), 1:4, sep = "_"),
  Condition = rep(c("A", "B", "D"), each = 4),
  Replicate = rep(1:4, times = 3),
  PC1 = c(-5.1, -4.6, -5.4, -4.9, 0.4, 0.9, 0.2, 0.7, 4.8, 5.2, 4.5, 5.0),
  PC2 = c(1.2, 0.7, -0.9, -1.1, 2.4, 1.8, -1.6, -2.2, 0.9, 0.4, -0.8, -1.3),
  PC1_Perc = 62.4,
  PC2_Perc = 18.7,
  Subset = "All proteins",
  stringsAsFactors = FALSE
)

hc <- pca_highchart(scores, group_order = c("A", "B", "D"),
  ellipse_type = "confidence", show_labels = TRUE)

class(hc)
```

pca_highchart_list *Build a List of PCA Plots for Multiple Subsets*

Description

Automatically builds PCA plots for "all", "any" and/or specific comparisons. Similar to factoextra::fviz_pca_ind with ellipse options.

Usage

```
pca_highchart_list(
  pca_input,
  modes = c("all", "any"),
  alpha = 0.05,
  color_by = "Condition",
  group_order = NULL,
  palette = NULL,
  addEllipses = TRUE,
  ellipse_type = c("convex", "confidence"),
  ellipse_level = 0.95,
  ellipse_fill_opacity = 0.12,
  ellipse_line_width = 1,
  ellipse_npoints = 100,
  point_size = 5,
  show_labels = FALSE,
  label_size = 10,
  center = TRUE,
  scale. = TRUE,
  filter_samples_to_comparison = FALSE
)
```

Arguments

pca_input	Data frame in long format (see build_pca_scores for the structure)
modes	Vector of modes to generate: "all", "any", and/or comparison names (default: c("all", "any"))
alpha	Significance threshold for DEPs (default: 0.05)
color_by	Column used to colour the points (default: "Condition")
group_order	Order of the groups/conditions (optional)
palette	Named vector of colours, or NULL for an automatic palette
addEllipses	Show ellipses/hulls around the groups (default: TRUE)
ellipse_type	Ellipse type: "convex" or "confidence" (default: "convex")
ellipse_level	Confidence level for ellipse_type = "confidence" (default: 0.95)
ellipse_fill_opacity	Ellipse fill opacity (0-1, default: 0.12)

ellipse_line_width	Outline line width (default: 1)
ellipse_npoints	Number of points for the confidence ellipse (default: 100)
point_size	Point radius (default: 5)
show_labels	Show the point labels (SampleID) (default: FALSE)
label_size	Label font size in px (default: 10)
center	Center the data before PCA (default: TRUE)
scale.	Scale the data before PCA (default: TRUE)
filter_samples_to_comparison	For specific comparisons, restrict the samples to the conditions involved (default: FALSE)

Value

Named list of highchart objects

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# One PCA over all proteins and one over the significant ones
hc_pcas <- pca_highchart_list(
  res$PCA_Input,
  modes      = c("all", "any"),
  group_order = c("A", "B", "D"),
  ellipse_type = "confidence",
  ellipse_level = 0.95
)
names(hc_pcas)
```

prepare_heatmap_data *Prepare Long-Format Data for tidyHeatmap*

Description

Prepare Long-Format Data for tidyHeatmap

Usage

```
prepare_heatmap_data(
  data,
  mode = c("all", "any", "target"),
  alpha = 0.05,
  comparison = NULL,
```

```
feature_ids = NULL,  
scale_data = c("row", "none", "column"),  
sample_order = c("clustering", "condition", "custom"),  
condition_order = NULL  
)
```

Arguments

data	Data frame in long format with columns: • SampleID: sample identifier • FeatureID: protein/feature identifier • Intensity: intensity value (log2) • Condition: experimental condition • Replicate: replicate number • sig_any: logical flagging significance (for mode="any") • adjP_*: adjusted p-value columns (for mode="target")
mode	Filtering mode: "all", "any", or "target"
alpha	Significance threshold for mode "target" (default: 0.05)
comparison	Name of the comparison for mode "target" (e.g. "B-A")
feature_ids	Vector of specific FeatureIDs (when supplied, mode/alpha are ignored)
scale_data	Scaling type: "none", "row", "column" (default: "row")
sample_order	Sample order: "clustering", "condition", or a custom vector
condition_order	Condition order when sample_order = "condition"

Value

Data frame in long format, ready for tidyHeatmap

preprocess_diann	<i>Preprocess DIA-NN protein-group matrices</i>
------------------	---

Description

Converts a DIA-NN protein-group matrix (report.pg_matrix.tsv, wide TSV format) into the same output structure as preprocess_spectronaut(): three data.frames (metadata, protein_id, protein_quant) ready for the downstream pipeline ([process_proteomics\(\)](#) and the associated modules).

DIA-NN names its intensity columns after the raw file, which says nothing about the experimental design, so the design has to be declared. There are two ways: rename the columns to Abundance: <Condition>_<Replicate> as Proteome Discoverer writes them, or pass a sample sheet through annot_path. The design is never inferred from the raw-file name – a guess that happened to be right for one export would be wrong and silent for the next.

A protein-group matrix carries only intensities. The identification metrics the wide contract expects are filled from what DIA-NN does report (N.Sequences, N.Proteotypic.Sequences), left as NA where it reports nothing (the precursor counts), or omitted where nothing downstream needs them (coverage and the Spectronaut C-score).

Usage

```
preprocess_diann(
  file_path,
  condition_order,
  annot_path = NULL,
  export_dir = NULL,
  timestamp_suffix = TRUE,
  verbose = TRUE
)
```

Arguments

<code>file_path</code>	Path to the <code>report.pg_matrix.tsv</code> exported by DIA-NN.
<code>condition_order</code>	Character vector with the order of the experimental conditions (e.g. <code>c("A", "B", "D")</code>). Only the runs whose condition is in this vector are kept, so it doubles as a filter. Conditions listed but absent from the report are dropped, with a warning, so that no empty factor level reaches the metadata.
<code>annot_path</code>	Path to a sample sheet with columns <code>Column</code> and <code>Condition</code> , used to read the report exactly as DIA-NN wrote it: <code>Column</code> holds the run headers, raw-file names and all. If <code>NULL</code> (default), the columns are expected to follow the Abundance: convention. Any other column in the sheet is ignored – batch structure belongs in <code>covariate_df</code> of <code>process_proteomics()</code> , see <code>vignette("batch-correction")</code> . The sheet is a file rather than a vector of labels because a vector would be positional, and nothing guarantees the column order of an export: a re-export that reordered them would relabel every run without any way to notice. Matching by name either matches or fails.
<code>export_dir</code>	Directory to export the TSV files to. If <code>NULL</code> (default), no files are exported.
<code>timestamp_suffix</code>	Logical. If <code>TRUE</code> (default), appends a timestamp to the names of the exported files.
<code>verbose</code>	Logical. If <code>TRUE</code> (default), shows progress messages.

Value

A list with class `c("diann_data", "proteomics_data", "list")` containing:

metadata Data frame with one record per run, including the number of protein groups quantified in each

protein_id Data frame with the identification metrics per protein (wide format, the global metrics replicated per run)

protein_quant Data frame with the quantification metrics per protein (includes `PG.Quantity_<Coding>`)

The call that produced the object and a fingerprint of the file it was read from (path, size, modification time and MD5) travel with it as the attributes `nadia_call` and `nadia_source`. They are attributes rather than list elements so that the three-element structure above is unchanged; `write_nadia()` records them as the provenance of an analysis.

Examples

```
# A trimmed DIA-NN protein-group matrix ships with the package. Its run
# columns have been renamed to the "Abundance: <Condition>_<Replicate>"
# convention, so no further argument is needed.
report <- system.file("extdata", "nadia_diann_report.tsv.gz",
                      package = "NADIA")

diann <- preprocess_diann(report, condition_order = c("A", "B", "D"),
                          verbose = FALSE)

diann
table(diann$metadata$R.Condition)

# Straight from DIA-NN, where the columns are still raw-file paths, the
# design is given instead by a sheet whose 'Column' holds those paths:
# preprocess_diann("report.pg_matrix.tsv",
#                  condition_order = c("A", "B", "D"),
#                  annot_path = "design.tsv")
```

```
preprocess_lfq
```

```
Preprocess Proteome Discoverer LFQ exports
```

Description

Converts a Proteome Discoverer LFQ protein export (wide TSV format) into the same output structure as `preprocess_spectronaut()` / `preprocess_tmt()`: three data.frames (metadata, protein_id, protein_quant) ready for the downstream pipeline (`process_proteomics()`).

The experimental design comes from the Abundance: <Condition>_<Replicate> column suffixes, exactly as in `preprocess_tmt()` and `preprocess_diann()`. For a report whose columns are not named that way, `annot_path` takes a sample sheet instead.

The intensities (Abundance: <sample>) and the three per-sample metric families (Score Mascot, # PSMs (by Search Engine), # Peptides (by Search Engine)) are mapped BY SAMPLE NAME, never by position: Proteome Discoverer writes the four families in different orders.

Usage

```
preprocess_lfq(
  file_path,
  annot_path = NULL,
  condition_order = NULL,
  export_dir = NULL,
  timestamp_suffix = TRUE,
```

```

    verbose = TRUE
  )

```

Arguments

<code>file_path</code>	Path to the data TSV exported from Proteome Discoverer.
<code>annot_path</code>	Path to a sample sheet with columns Column and Condition, where Column holds the text after Abundance: . If NULL (default), the design comes from the suffixes. Any other column in the sheet is ignored – batch structure belongs in <code>covariate_df</code> of <code>process_proteomics()</code> , see <code>vignette("batch-correction")</code> .
<code>condition_order</code>	Character vector with the order of the conditions. If NULL (default), it is derived from the data (order of appearance). If supplied, it fixes the factor levels and discards samples whose condition is not in the list; conditions listed but absent are dropped too, with a warning, so that no empty factor level reaches the metadata.
<code>export_dir</code>	Directory to export the TSVs to. NULL (default) = no export.
<code>timestamp_suffix</code>	Logical. If TRUE (default), appends a timestamp to the exported files.
<code>verbose</code>	Logical. If TRUE (default), shows progress messages.

Value

A list with class `c("lfq_data", "proteomics_data", "list")` containing `metadata`, `protein_id` and `protein_quant`.

The call that produced the object and a fingerprint of the files it was read from – the report, and the sample sheet when there is one – (path, size, modification time and MD5) travel with it as the attributes `nadia_call` and `nadia_source`. They are attributes rather than list elements so that the three-element structure above is unchanged; `write_nadia()` records them as the provenance of an analysis.

Examples

```

# A trimmed Proteome Discoverer LFQ report ships with the package. Its columns
# follow the "Abundance: <Condition>_<Replicate>" convention, so the design
# needs no further argument.
report <- system.file("extdata", "nadia_lfq_report.tsv.gz", package = "NADIA")

lfq <- preprocess_lfq(report, verbose = FALSE)
lfq
lfq$metadata[, c("Coding", "R.Condition", "R.Replicate")]

# The same design, declared in a sheet instead. Use this route when the
# columns are not named by the convention; here it is the same experiment, so
# it produces the same sample table.
annot <- system.file("extdata", "nadia_lfq_annotation.tsv", package = "NADIA")
from_sheet <- preprocess_lfq(report, annot_path = annot, verbose = FALSE)
identical(from_sheet$metadata, lfq$metadata)

```

preprocess_spectronaut

Preprocess Spectronaut reports

Description

Converts a Spectronaut report (long TSV format) into three structured tables: run metadata, protein identification and protein quantification.

The report has to carry the right columns, and rather than leave that to be reconstructed by hand the package ships the Spectronaut export schema that produces them:

```
system.file("extdata", "NADIA_Report.rs", package = "NADIA")
```

Import that file into Spectronaut as a report schema and export with it. The result has the shape of `nadia_dia_report.tsv.gz`, the example that ships alongside it.

Usage

```
preprocess_spectronaut(
  file_path,
  condition_order,
  export_dir = NULL,
  agg_coverage_run = c("max", "mean", "median", "min"),
  agg_coverage_global = c("max", "mean", "median", "min"),
  agg_mw = c("max", "mean", "median", "min"),
  agg_cscore_runwise = c("mean", "max", "median", "min"),
  timestamp_suffix = TRUE,
  verbose = TRUE
)
```

Arguments

<code>file_path</code>	Path to the Spectronaut TSV file.
<code>condition_order</code>	Character vector that selects the experimental conditions to retain and sets their order in subsequent analysis stages (e.g. <code>c("Control", "Treated")</code>). Runs whose condition is not listed are dropped; conditions listed but absent from the report are dropped too, with a warning, so that no empty factor level reaches the metadata.
<code>export_dir</code>	Directory to export the TSV files to. If NULL (default), no files are exported.
<code>agg_coverage_run</code>	Aggregation method for the per-sample PG.Coverage. Options: "max" (default), "mean", "median", "min".
<code>agg_coverage_global</code>	Aggregation method for PG.Coverage.Global. Options: "max" (default), "mean", "median", "min".

agg_mw	Aggregation method for PG.MolecularWeight. Options: "max" (default), "mean", "median", "min".
agg_cscore_runwise	Aggregation method for PG.Cscore.RunWise. Options: "mean" (default), "max", "median", "min".
timestamp_suffix	Logical. If TRUE (default), appends a timestamp to the names of the exported files.
verbose	Logical. If TRUE (default), shows progress messages.

Value

A list with class `spectronaut_data` containing:

metadata Data frame with the run information (1 row per sample)

protein_id Data frame with the identification metrics per protein

protein_quant Data frame with the quantification metrics per protein

The call that produced the object and a fingerprint of the file it was read from (path, size, modification time and MD5) travel with it as the attributes `nadia_call` and `nadia_source`. They are attributes rather than list elements so that the three-element structure above is unchanged; `write_nadia()` records them as the provenance of an analysis.

Examples

```
# A trimmed Spectronaut report ships with the package
report <- system.file("extdata", "nadia_dia_report.tsv.gz", package = "NADIA")

prep <- preprocess_spectronaut(report,
                              condition_order = c("A", "B", "D"),
                              verbose = FALSE)

prep
dim(prep$protein_quant)

# Nothing is written to disk unless export_dir is given
prep <- preprocess_spectronaut(report,
                              condition_order = c("A", "B", "D"),
                              export_dir = tempdir(),
                              verbose = FALSE)
```

Description

Converts a Proteome Discoverer protein export (wide TSV format, with Abundance: <Condition>_<Replicate> columns) into the same output structure as `preprocess_spectronaut()`: three data.frames (metadata, protein_id, protein_quant) ready for the downstream pipeline (`process_proteomics()` and the associated modules).

The global PD identification metrics (# PSMs, # Peptides, # Unique Peptides, Coverage [%]) are replicated into per-channel columns to fit the wide Spectronaut contract.

Usage

```
preprocess_tmt(
  file_path,
  condition_order,
  annot_path = NULL,
  export_dir = NULL,
  timestamp_suffix = TRUE,
  verbose = TRUE
)
```

Arguments

<code>file_path</code>	Path to the TSV exported from Proteome Discoverer.
<code>condition_order</code>	Character vector with the order of the experimental conditions (e.g. <code>c("A", "B", "D", "IS")</code>). Only the channels whose condition is in this vector are kept – handy for excluding Internal Standards by omitting "IS". Conditions listed but absent from the report are dropped, with a warning, so that no empty factor level reaches the metadata.
<code>annot_path</code>	Path to a sample sheet with columns Column and Condition, for reports whose channels are not named <Condition>_<Replicate> – an export left with the TMT tags, 126, 127N and so on. Column holds the text after Abundance: . If NULL (default), the design comes from the suffixes. Any other column in the sheet is ignored – batch structure belongs in <code>covariate_df</code> of <code>process_proteomics()</code> , see <code>vignette("batch-correction")</code> .
<code>export_dir</code>	Directory to export the TSV files to. If NULL (default), no files are exported.
<code>timestamp_suffix</code>	Logical. If TRUE (default), appends a timestamp to the names of the exported files.
<code>verbose</code>	Logical. If TRUE (default), shows progress messages.

Value

A list with class `c("tmt_data", "proteomics_data", "list")` containing:

metadata Data frame with one record per channel/sample

protein_id Data frame with the identification metrics per protein (wide format, global metrics replicated per channel)

protein_quant Data frame with the quantification metrics per protein (includes PG.Quantity_<Coding>)

The call that produced the object and a fingerprint of the file it was read from (path, size, modification time and MD5) travel with it as the attributes `nadia_call` and `nadia_source`. They are attributes rather than list elements so that the three-element structure above is unchanged; `write_nadia()` records them as the provenance of an analysis.

Examples

```
# A trimmed Proteome Discoverer TMTpro report ships with the package: three
# conditions of eight replicates, split across two TMT mixes (replicates 1-4
# are the first mix, 5-8 the second) and bridged by four "IS" channels.
report <- system.file("extdata", "nadia_tmt_report.tsv.gz", package = "NADIA")

# condition_order is required, and doubles as a filter: the report also holds
# an "IS" channel (internal standards) that is left out by not listing it
tmt <- preprocess_tmt(report,
                      condition_order = c("A", "B", "D"),
                      verbose = FALSE)

tmt
table(tmt$metadata$R.Condition)
```

print.diann_data	<i>Print a summary of a DIA-NN preprocessing result</i>
------------------	---

Description

Reports the number of runs, protein groups and conditions, so that the shape of the experiment can be checked at a glance before processing it.

Usage

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

Arguments

<code>x</code>	A <code>diann_data</code> object, as returned by <code>preprocess_diann()</code> .
<code>...</code>	Ignored, present for compatibility with the <code>print</code> generic.

Value

`x`, invisibly. Called for the summary it prints.

Examples

```
diann <- preprocess_diann(
  system.file("extdata", "nadia_diann_report.tsv.gz", package = "NADIA"),
  condition_order = c("A", "B", "D"),
  verbose = FALSE)
print(diann)
```

print.lfq_data	<i>Print a summary of an LFQ preprocessing result</i>
----------------	---

Description

Reports the number of samples, protein groups and conditions, so that the shape of the experiment can be checked at a glance before processing it.

Usage

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

Arguments

x	An lfq_data object, as returned by preprocess_lfq() .
...	Ignored, present for compatibility with the print generic.

Value

x, invisibly. Called for the summary it prints.

Examples

```
lfq <- preprocess_lfq(
  system.file("extdata", "nadia_lfq_report.tsv.gz", package = "NADIA"),
  annot_path = system.file("extdata", "nadia_lfq_annotation.tsv",
                           package = "NADIA"),
  verbose = FALSE)
print(lfq)
```

print.nadia_db	<i>Print a summary of a .nadia file</i>
----------------	---

Description

Print a summary of a .nadia file

Usage

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

Arguments

x	A nadia_db object.
...	Ignored.

Value

x, invisibly.

print.proteomics_heatmap	<i>Draw a proteomics heatmap</i>
--------------------------	----------------------------------

Description

Renders the object built by [proteomics_heatmap\(\)](#) on the current graphics device, adding the title stored in the object. Printing is what actually draws the heatmap: building it does not.

Usage

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

Arguments

x	A proteomics_heatmap object, as returned by proteomics_heatmap() .
...	Further arguments passed to <code>ComplexHeatmap::draw()</code> .

Value

The drawn `HeatmapList`, invisibly. Called for its side effect.

Examples

```

if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("tidyHeatmap", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  hm <- proteomics_heatmap(res$PCA_Input, mode = "any")

  # Drawing needs a graphics device; send it to a temporary file
  f <- tempfile(fileext = ".png")
  grDevices::png(f)
  print(hm)
  grDevices::dev.off()
  file.remove(f)
}

```

```
print.proteomics_result
```

Print method for proteomics_result

Description

Print method for proteomics_result

Usage

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

Arguments

<code>x</code>	proteomics_result object
<code>...</code>	Additional arguments (ignored)

Value

`x`, invisibly. Called for the summary it prints: the assays of the SummarizedExperiment, the comparisons performed, the number of differential proteins and the parameters used.

Examples

```

data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)
print(res)

```

```
print.spectronaut_data
```

Print a summary of a Spectronaut preprocessing result

Description

Reports the number of runs, protein groups and conditions, so that the shape of the experiment can be checked at a glance before processing it.

Usage

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

Arguments

x	A spectronaut_data object, as returned by preprocess_spectronaut() .
...	Ignored, present for compatibility with the print generic.

Value

x, invisibly. Called for the summary it prints.

Examples

```
data(nadia_dia)  
print(nadia_dia)
```

```
print.tmt_data
```

Print a summary of a TMT preprocessing result

Description

Reports the number of channels, protein groups and conditions, so that the shape of the experiment can be checked at a glance before processing it.

Usage

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

Arguments

x	A tmt_data object, as returned by preprocess_tmt() .
...	Ignored, present for compatibility with the print generic.

Value

x, invisibly. Called for the summary it prints.

Examples

```
tmt <- preprocess_tmt(
  system.file("extdata", "nadia_tmt_report.tsv.gz", package = "NADIA"),
  condition_order = c("A", "B", "D"),
  verbose = FALSE)
print(tmt)
```

process_proteomics	<i>Process proteomics data</i>
--------------------	--------------------------------

Description

Complete processing pipeline: filtering, normalization, imputation and differential expression analysis. Coordinates sub-modules Normalization.R, Imputation.R, and DEAnalysis.R.

Usage

```
process_proteomics(
  preprocessing,
  export_dir = NULL,
  min_reps_filter = NULL,
  min_groups_filter = 1,
  norm_method = "cycloess",
  cyclic_loess_method = "fast",
  cyclic_loess_iterations = 3,
  cyclic_loess_span = 0.7,
  batch_correct = FALSE,
  batch_column = "Batch",
  batch_algorithm = "ComBat",
  batch_ComBat_mode = 1,
  batch_covariates = NULL,
  batch_qualitycontrol = FALSE,
  imp_method = "combo",
  mar_method = "Impseqrob",
  mnar_method = "min",
  prop_na_mnar = 0.51,
  prop_present_mar = 0.5,
  min_present_mar = 1,
  require_n_conditions = 1,
  max_na_prop = NULL,
  method_args = list(),
  with_value = NA_real_,
```

```

    comparisons = NULL,
    control = NULL,
    logFC_threshold = 0,
    alpha = 0.05,
    eBayes_trend = NULL,
    eBayes_robust = NULL,
    de_method = "limma",
    covariate_df = NULL,
    covariate_column = NULL,
    bio_replicate_column = NULL,
    export_normalized = TRUE,
    export_imputed = TRUE,
    export_format = "tsv",
    export_volcano = TRUE,
    export_boxplot = TRUE,
    export_pca = TRUE,
    verbose = TRUE
)

```

Arguments

- preprocessing** A proteomics_data object, as returned by [preprocess_spectronaut\(\)](#), [preprocess_diann\(\)](#), [preprocess_tmt\(\)](#) or [preprocess_lfq\(\)](#).
- export_dir** Output directory for exported files. Defaults to NULL, which writes nothing to disk; pass a path to enable the exports controlled by export_normalized, export_imputed, export_volcano, export_boxplot and export_pca.
- min_reps_filter** Minimum replicates for filtering. If NULL, auto-computed
- min_groups_filter** Minimum groups for filtering (default: 1)
- norm_method** Normalization method passed to normalize_proteomics() (default: "cycloess"). See normalize_proteomics() for all 13 options.
- cyclic_loess_method** Cyclic Loess method: "fast" or "pairs" (default: "fast")
- cyclic_loess_iterations** Number of iterations for Cyclic Loess (default: 3)
- cyclic_loess_span** Span parameter for Cyclic Loess (default: 0.7)
- batch_correct** Logical. Apply BERT batch correction after normalization (default: FALSE). Requires covariate_df with batch_column.
- batch_column** Column in covariate_df containing batch assignments (default: "Batch"). Must have >= 2 unique values.
- batch_algorithm** Batch correction algorithm: "ComBat" (default), "limma", or "ref"
- batch_ComBat_mode** Integer 1-4 for ComBat parametric/mean-only settings (default: 1)

batch_covariates	Character vector of colData column names to protect during BERT batch correction (default: NULL). Categorical columns are encoded as indicator variables automatically; numeric columns are passed through unchanged and therefore act as continuous covariates.
batch_qualitycontrol	Logical. Compute ASW quality metrics (default: FALSE)
imp_method	Imputation method (default: "combo"). See impute_proteomics() for all options.
mar_method	MAR method for combo mode (default: "Impseqrob")
mnar_method	MNAR method for combo mode (default: "min")
prop_na_mnar	NA proportion for MNAR classification (default: 0.51)
prop_present_mar	Present proportion for MAR (default: 0.5)
min_present_mar	Minimum present values for MAR (default: 1)
require_n_conditions	Required conditions with presence (default: 1)
max_na_prop	Maximum NA proportion above which a protein is removed before imputation. NULL (the default) disables the filter.
method_args	Named list of per-method argument lists
with_value	Constant value for imp_method="with"
comparisons	Comparisons for DE. If NULL, generates all pairwise
control	Control condition. If NULL, compares all
logFC_threshold	LogFC threshold for significance (default: 0)
alpha	Adjusted p-value threshold (default: 0.05)
eBayes_trend	Use trend estimation in eBayes. If NULL (default), it is resolved from de_method: TRUE for "limma", FALSE for "limpa".
eBayes_robust	Use robust estimation in eBayes. If NULL (default), it is resolved from de_method: TRUE for "limma", FALSE for "limpa".
de_method	DE method: "limma" (default) or "limpa" (probabilistic, requires imp_method="limpa")
covariate_df	Data frame with Column + covariate column(s) for paired/blocked design (default: NULL)
covariate_column	Name(s) of the covariate column(s) for the DE model. Single string (e.g., "Subject") or character vector (e.g., c("Subject", "Batch")). Default: NULL
bio_replicate_column	Column name in covariate_df identifying biological replicates (e.g., "Patient", "Subject"). Used with limma::duplicateCorrelation() to account for technical replicates or paired designs via random effect blocking. Default: NULL
export_normalized	Export normalized matrix (default: TRUE)
export_imputed	Export imputed matrix (default: TRUE)

```

export_format  Export format: "tsv", "parquet", or "both" (default: "tsv")
export_volcano Export VolcanoPlot_Input (default: TRUE)
export_boxplot Export BoxPlot_Input (default: TRUE)
export_pca     Export PCA_Input (default: TRUE)
verbose        Print progress messages (default: TRUE)

```

Details

The input is a `proteomics_data` object, which every preprocessing function returns with the same three elements, so the pipeline is indifferent to the acquisition and the search engine the data came from.

Value

List of class `proteomics_result` with:

- `se_proc`: Processed SummarizedExperiment with all assays
- `DEPs_results`: Data frame with differential expression results
- `BoxPlot_Input`: Long-format data frame with one row per protein, sample and assay (columns Column, Assay, Intensity, Condition, Replicate, Protein.IDs). Feed it to [boxplot_highchart_list\(\)](#).
- `PCA_Input`: Long-format data frame with the imputed intensities plus one `adjP_*` column per comparison and a `sig_any` flag. Feed it to [pca_highchart_list\(\)](#), [proteomics_heatmap\(\)](#) or [proteomics_heatmap_list\(\)](#).
- `comparisons`: Comparisons performed
- `parameters`: Parameters used

`BoxPlot_Input` and `PCA_Input` are always returned, whether or not `export_dir` is given: the plotting layer needs them, and requiring a round trip through disk to obtain them would be gratuitous.

Examples

```

data(nadia_dia)

# Normalization, imputation and differential expression in one call
res <- process_proteomics(nadia_dia, verbose = FALSE)
res

SummarizedExperiment::assayNames(res$se_proc)
table(res$DEPs_results$Comparison, res$DEPs_results$Change)

# The plotting layer is fed straight from the result, no files involved
names(res$BoxPlot_Input)
names(res$PCA_Input)

# Another combination: quantile normalization and a single imputation method
res2 <- process_proteomics(nadia_dia, norm_method = "quantile",
                           imp_method = "QRILC", verbose = FALSE)
SummarizedExperiment::assayNames(res2$se_proc)

```

protein_list_reactable

Interactive Reactable Table for Protein_ID Data (post-Spectronaut)

Description

Builds a reactable table with 2-level grouped headers, static columns pinned (sticky) to the left and data bars coloured by condition inside each numeric cell. It reproduces the usual Excel layout and improves on it.

Usage

```
protein_list_reactable(
  data,
  metadata = NULL,
  page_size = 15,
  height = 720,
  element_id = NULL,
  selection = NULL,
  searchable = TRUE
)
```

Arguments

data	Data frame or path to a Protein_ID TSV/CSV/Parquet file (output of any pre-process_*() function -> protein_id).
metadata	Optional: metadata data frame (run_summary) with a Coding column that fixes the order of the sample columns.
page_size	Rows per page (default 15).
height	Table height in px (default 720).
element_id	Reactable element ID (default NULL).
selection	"single" "multiple" NULL.
searchable	Enable the search box (default TRUE).

Value

A reactable object.

Examples

```
data(nadia_dia)

# metadata$Coding fixes the order of the sample columns
tbl <- protein_list_reactable(nadia_dia$protein_id,
                             metadata = nadia_dia$metadata)

class(tbl)
```

protein_list_widget	<i>Full Widget for the Protein_ID Table with Filters, Search and Excel Export</i>
---------------------	---

Description

Wraps protein_list_reactable() with:

- A search bar and buttons (toggle filters, clear, export to Excel).
- Per-condition chips that hide/show the 16 associated columns.
- A numeric filter on MW in Da with AND/OR operators.
- Excel export with 2-level headers (merged groups + sub-labels) and a per-condition tint, reproducing the manual layout.

Usage

```
protein_list_widget(  
  data,  
  metadata = NULL,  
  page_size = 15,  
  height = 720,  
  element_id = "protein_id_table",  
  selection = NULL,  
  searchable = TRUE  
)
```

Arguments

data	Data frame or path to a Protein_ID TSV/CSV/Parquet file (output of any pre-process_*() function -> protein_id).
metadata	Optional: metadata data frame (run_summary) with a Coding column that fixes the order of the sample columns.
page_size	Rows per page (default 15).
height	Table height in px (default 720).
element_id	Element ID (default "protein_id_table").
selection	"single" "multiple" NULL.
searchable	Enable the search box (default TRUE).

Value

A browsable htmltools object.

Examples

```

if (requireNamespace("jsonlite", quietly = TRUE)) {
  data(nadia_dia)

  w <- protein_list_widget(nadia_dia$protein_id,
                           metadata = nadia_dia$metadata)
  print(class(w)) # print(w) itself opens it in the Viewer / browser
}

```

proteomics_heatmap	<i>Create a tidyHeatmap Heatmap for Proteomics</i>
--------------------	--

Description

Create a tidyHeatmap Heatmap for Proteomics

Usage

```

proteomics_heatmap(
  data,
  mode = c("all", "any", "target"),
  alpha = 0.05,
  comparison = NULL,
  feature_ids = NULL,
  row_annotation = NULL,
  row_annotation_cols = NULL,
  row_annotation_palette = NULL,
  row_annotation_size = NULL,
  row_annotation_name_size = 8,
  row_order_by = NULL,
  split_rows_by = NULL,
  scale_data = c("row", "none", "column"),
  sample_order = "clustering",
  condition_order = NULL,
  cluster_rows = FALSE,
  cluster_columns = TRUE,
  show_row_names = NULL,
  show_column_names = TRUE,
  palette_value = NULL,
  palette_annotation = NULL,
  reverse_palette = FALSE,
  row_title = "Proteins",
  column_title = "Samples",
  row_title_size = 10,
  column_title_size = 10,
  show_row_title = TRUE,

```

```

    show_column_title = TRUE,
    show_annotation = TRUE,
    split_by_condition = FALSE,
    show_adjp_annotation = TRUE,
    palette_adjp = NULL,
    row_names_size = 7,
    column_names_size = 9,
    column_names_rotation = 45,
    column_dend_height = NULL,
    row_dend_width = NULL,
    show_heatmap_legend = TRUE,
    show_annotation_legend = TRUE,
    border_color = NULL,
    heatmap_title = NULL,
    heatmap_title_size = 14,
    heatmap_title_face = c("bold", "plain", "italic", "bold.italic"),
    export_path = NULL,
    export_file = NULL,
    plot_width = 10,
    plot_height = 8,
    export_dpi = 300
  )

```

Arguments

<code>data</code>	Data frame in long format (see <code>prepare_heatmap_data</code> for the structure)
<code>mode</code>	Protein filtering mode: "all", "any", or "target"
<code>alpha</code>	Significance threshold for DEP proteins (default: 0.05)
<code>comparison</code>	Name of the comparison for mode "target" (e.g. "B-A")
<code>feature_ids</code>	Vector of specific FeatureIDs to show (default: NULL, uses the mode-based filtering). When supplied, only these proteins are shown and the mode/alpha filtering is ignored.
<code>row_annotation</code>	Row annotations (FeatureIDs). It can be: • Path to a TSV file with a "FeatureID" column plus additional categorical columns • Data frame with the same structure • NULL: no row annotations (default)
<code>row_annotation_cols</code>	Vector of column names to display as annotations. Default: NULL (uses every column except FeatureID)
<code>row_annotation_palette</code>	Named list of palettes, one per annotation. Example: <code>list(Pathway = "brewer:Set1", Function = c("red", "blue", "green"))</code>
<code>row_annotation_size</code>	Width of the row annotation bars. It can be: • Number: interpreted as centimetres (e.g. 0.3 = 0.3cm)

	• A unit object: <code>grid::unit(0.3, "cm")</code> • NULL: use <code>tidyHeatmap</code>'s default value
<code>row_annotation_name_size</code>	Font size of the row annotation names (default: 8)
<code>row_order_by</code>	Row order. It can be: • NULL: no particular ordering (default) • "clustering": order by hierarchical clustering • A column name: order by that annotation (e.g. "Specie") • A vector of columns: order sequentially (e.g. <code>c("Specie", "Process", "Function")</code>) • Include "adjP" when <code>mode = "target"</code> to order by adjusted p-value
<code>split_rows_by</code>	Name of the annotation column used to split the rows into groups (default: NULL)
<code>scale_data</code>	Scaling type: "none", "row", "column" (default: "row")
<code>sample_order</code>	Sample order: "clustering", "condition", or a custom vector
<code>condition_order</code>	Condition order when <code>sample_order = "condition"</code>
<code>cluster_rows</code>	Cluster the rows (default: FALSE)
<code>cluster_columns</code>	Cluster the columns (default: TRUE)
<code>show_row_names</code>	Show the row names (default: TRUE when ≤ 50 proteins)
<code>show_column_names</code>	Show the column names (default: TRUE)
<code>palette_value</code>	Palette for the heatmap values (see <code>get_heatmap_palette</code>)
<code>palette_annotation</code>	Palette for the Condition annotation
<code>reverse_palette</code>	Reverse the value palette (default: FALSE)
<code>row_title</code>	Title for the rows
<code>column_title</code>	Title for the columns
<code>row_title_size</code>	Font size of the row title (default: 10)
<code>column_title_size</code>	Font size of the column title (default: 10)
<code>show_row_title</code>	Show the row title (default: TRUE)
<code>show_column_title</code>	Show the column title (default: TRUE)
<code>show_annotation</code>	Show the Condition annotation (default: TRUE)
<code>split_by_condition</code>	Split the heatmap by condition (default: FALSE)
<code>show_adjP_annotation</code>	Show the adjP row annotation when <code>mode="target"</code> (default: TRUE)

palette_adjp	Palette for the adjP annotation. It can be: • NULL: use the default palette (red-orange-white) • A vector of 3 colours: <code>c(color_0, color_middle, color_0.05)</code> • String "brewer:name": use an RColorBrewer palette • String "package::palette": use a paletteer palette
row_names_size	Font size of the row names (default: 7)
column_names_size	Font size of the column names (default: 9)
column_names_rotation	Rotation of the column names in degrees (default: 45)
column_dend_height	Height of the column dendrogram. It can be: • Number: interpreted as millimetres (e.g. 30 = 30mm) • A unit object: <code>grid::unit(2, "cm")</code> • NULL: use ComplexHeatmap's default value
row_dend_width	Width of the row dendrogram. Same format as column_dend_height
show_heatmap_legend	Show the heatmap legend (default: TRUE)
show_annotation_legend	Show the annotation legends (default: TRUE)
border_color	Border colour of the heatmap cells. It can be: • NULL or FALSE: no border (default) • TRUE: black border • A colour string: that specific colour (e.g. "black", "grey", "#CCCCCC")
heatmap_title	Main heatmap title (default: NULL, no title)
heatmap_title_size	Font size of the main title (default: 14)
heatmap_title_face	Font face of the title: "plain", "bold", "italic", "bold.italic" (default: "bold")
export_path	Path used to export the heatmap data to TSV (default: NULL, nothing is exported). The file contains FeatureID, the per-sample intensity values, and the metadata (adjP where applicable).
export_file	Path used to export the plot. The format is taken from the extension: • .png: PNG image (raster) • .svg: SVG image (vector) • .pdf: PDF document (vector)
plot_width	Plot width in inches (default: 10). To get pixels: $\text{pixels} = \text{inches} \times \text{dpi}$ (e.g. 10" x 300dpi = 3000px)
plot_height	Plot height in inches (default: 8). To get pixels: $\text{pixels} = \text{inches} \times \text{dpi}$ (e.g. 8" x 300dpi = 2400px)
export_dpi	Resolution for PNG output in dots per inch (default: 300). Higher dpi = more detail. Common values: 72 (web), 150 (draft), 300 (publication)

Value

tidyHeatmap/ComplexHeatmap object

Examples

```
if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("tidyHeatmap", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  # Proteins significant in the B-A comparison, samples grouped by condition
  hm <- proteomics_heatmap(
    res$PCA_Input,
    mode          = "target",
    comparison     = "B-A",
    scale_data     = "row",
    sample_order   = "condition",
    condition_order = c("A", "B", "D")
  )
  print(class(hm)) # print(hm) itself draws it on the current device
}
```

proteomics_heatmap_list

Build a List of Heatmaps for Several Subsets

Description

Automatically builds heatmaps for "all", "any" and/or specific comparisons.

Usage

```
proteomics_heatmap_list(
  data,
  modes = c("all", "any"),
  alpha = 0.05,
  feature_ids = NULL,
  row_annotation = NULL,
  row_annotation_cols = NULL,
  row_annotation_palette = NULL,
  row_annotation_size = NULL,
  row_annotation_name_size = 8,
  row_order_by = NULL,
  split_rows_by = NULL,
  scale_data = c("row", "none", "column"),
  sample_order = "clustering",
  condition_order = NULL,
```

```

cluster_rows = FALSE,
cluster_columns = TRUE,
show_row_names = NULL,
show_column_names = TRUE,
palette_value = NULL,
palette_annotation = NULL,
reverse_palette = FALSE,
row_title_size = 10,
column_title_size = 10,
show_row_title = TRUE,
show_column_title = TRUE,
show_annotation = TRUE,
split_by_condition = FALSE,
show_adj_p_annotation = TRUE,
palette_adj_p = NULL,
row_names_size = 7,
column_names_size = 9,
column_names_rotation = 45,
column_dend_height = NULL,
row_dend_width = NULL,
show_heatmap_legend = TRUE,
show_annotation_legend = TRUE,
border_color = NULL,
heatmap_title = NULL,
heatmap_title_size = 14,
heatmap_title_face = "bold",
export_path = NULL,
export_modes = NULL,
export_file = NULL,
plot_width = 10,
plot_height = 8,
export_dpi = 300
)

```

Arguments

<code>data</code>	Data frame in long format (see <code>prepare_heatmap_data</code> for the structure)
<code>modes</code>	Vector of modes to generate: "all", "any", and/or comparison names (default: <code>c("all", "any")</code>)
<code>alpha</code>	Significance threshold for DEP proteins (default: 0.05)
<code>feature_ids</code>	Vector of specific FeatureIDs to show (default: NULL)
<code>row_annotation</code>	Row annotations (TSV path or data.frame)
<code>row_annotation_cols</code>	Columns to use as row annotations
<code>row_annotation_palette</code>	List of palettes for the row annotations
<code>row_annotation_size</code>	Width of the row annotation bars (a number in cm, or a unit)

row_annotation_name_size Font size of the row annotation names (default: 8)
 row_order_by Row order: "clustering", annotation column(s), or include "adjP" for mode="target"
 split_rows_by Annotation column used to split the rows into groups
 scale_data Scaling type: "none", "row", "column" (default: "row")
 sample_order Sample order: "clustering", "condition", or a custom vector
 condition_order Condition order when sample_order = "condition"
 cluster_rows Cluster the rows (default: FALSE)
 cluster_columns Cluster the columns (default: TRUE)
 show_row_names Show the row names (default: auto)
 show_column_names Show the column names (default: TRUE)
 palette_value Palette for the heatmap values
 palette_annotation Palette for the Condition annotation
 reverse_palette Reverse the value palette (default: FALSE)
 row_title_size Font size of the row title (default: 10)
 column_title_size Font size of the column title (default: 10)
 show_row_title Show the row title (default: TRUE)
 show_column_title Show the column title (default: TRUE)
 show_annotation Show the Condition annotation (default: TRUE)
 split_by_condition Split the heatmap by condition (default: FALSE)
 show_adjP_annotation Show the adjP row annotation for comparisons (default: TRUE)
 palette_adjP Palette for the adjP annotation (see proteomics_heatmap)
 row_names_size Font size of the row names (default: 7)
 column_names_size Font size of the column names (default: 9)
 column_names_rotation Rotation of the column names (default: 45)
 column_dend_height Height of the column dendrogram (a number in mm, or a unit)
 row_dend_width Width of the row dendrogram (a number in mm, or a unit)
 show_heatmap_legend Show the heatmap legend (default: TRUE)

show_annotation_legend	Show the annotation legends (default: TRUE)
border_color	Border colour of the cells (NULL, TRUE, or a colour)
heatmap_title	Main heatmap title (default: NULL, no title). "{mode}" can be used as a placeholder, and is replaced by the mode name
heatmap_title_size	Font size of the main title (default: 14)
heatmap_title_face	Font face of the title (default: "bold")
export_path	Base path used to export the data to TSV (default: NULL). The mode name is appended to the file name (e.g. "export_all.tsv", "export_B-A.tsv")
export_modes	Vector of modes to export (default: NULL, exports all of them). Only relevant when export_path is set. Example: c("all", "B-A")
export_file	Base path used to export the plots. The mode is appended to the name. The format is taken from the extension (.png, .svg, .pdf)
plot_width	Plot width in inches (default: 10)
plot_height	Plot height in inches (default: 8)
export_dpi	Resolution for PNG output in dots per inch (default: 300)

Value

Named list of tidyHeatmap objects

Examples

```

if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("tidyHeatmap", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  # "any" plus one heatmap per named comparison
  hm_list <- proteomics_heatmap_list(
    res$PCA_Input,
    modes          = c("any", "B-A"),
    sample_order   = "condition",
    condition_order = c("A", "B", "D"),
    palette_value  = "brewer:RdYlBu"
  )
  print(names(hm_list))  # print(hm_list[["B-A"]]) draws one of them
}

```

pvca_analysis *PVCA Analysis – Orchestrator*

Description

Runs the full PVCA workflow: compute variance components, generate plot, and optionally export results.

Usage

```
pvca_analysis(
  se,
  assay_name = NULL,
  technical_factors = character(0),
  biological_factors = character(0),
  pca_covariates = NULL,
  pca_threshold = 0.6,
  variance_threshold = 0.01,
  na_action = "complete",
  fill_value = -1,
  colors = NULL,
  verbose = TRUE,
  output_dir = NULL,
  export_plots = TRUE,
  export_tables = TRUE,
  plot_width = 10,
  plot_height = 6,
  plot_dpi = 150
)
```

Arguments

<code>se</code>	SummarizedExperiment object.
<code>assay_name</code>	Character. Assay to analyze. NULL = second assay (normalized, pre-imputation).
<code>technical_factors</code>	Character vector. Factors representing technical variation (e.g., "Injection", "Digestion").
<code>biological_factors</code>	Character vector. Factors representing biological variation (e.g., "Condition", "Gender", "Age").
<code>pca_covariates</code>	Character vector or NULL. If provided, generates PCA plots colored by each covariate (grid + individual plots). These are included in the returned list and exported as PNGs. Default NULL (skip).
<code>pca_threshold</code>	Numeric (0-1). Cumulative variance threshold for PCA (default 0.6).
<code>variance_threshold</code>	Numeric (0-1). Minimum weight for individual display (default 0.01).

<code>na_action</code>	Character: "complete", "fill", or "none" (default "complete").
<code>fill_value</code>	Numeric. NA replacement when <code>na_action = "fill"</code> (default -1).
<code>colors</code>	Named character vector for category colors (default NULL = built-in palette).
<code>verbose</code>	Logical (default TRUE).
<code>output_dir</code>	Character. Directory for exports. NULL = no export.
<code>export_plots</code>	Logical (default TRUE).
<code>export_tables</code>	Logical (default TRUE).
<code>plot_width</code>	Numeric in inches (default 10).
<code>plot_height</code>	Numeric in inches (default 6).
<code>plot_dpi</code>	Numeric (default 150).

Value

Named list:

variance_components data.frame with label, weights, category

plot ggplot2 object

n_proteins Number of proteins used in the analysis

assay_name Assay analyzed

parameters List of parameters used

Examples

```
data(nadia_dia)

# nadia_dia has no batch information, so the example builds a plausible one:
# two digestion batches crossed with the three conditions.
cov <- data.frame(Column = nadia_dia$metadata$Coding,
                  Batch = rep(c("b1", "b2"),
                             length.out = nrow(nadia_dia$metadata)),
                  stringsAsFactors = FALSE)
res <- process_proteomics(nadia_dia, covariate_df = cov, verbose = FALSE)

# Variance decomposition: how much of the signal each factor explains
pv <- pvca_analysis(res$se_proc, assay_name = "Impseqrob_min",
                   technical_factors = "Batch",
                   biological_factors = "Condition",
                   verbose = FALSE)

pv$pvca_table
```

pvca_compute

*Compute PVCA variance components***Description**

Performs Principal Variance Component Analysis on a SummarizedExperiment object. Decomposes total variance into contributions from each specified factor, their pairwise interactions, and residual variance.

Usage

```
pvca_compute(
  se,
  assay_name = NULL,
  factors,
  pca_threshold = 0.6,
  variance_threshold = 0.01,
  na_action = "complete",
  fill_value = -1,
  verbose = TRUE
)
```

Arguments

se	SummarizedExperiment object.
assay_name	Character. Name of the assay to use. If NULL, uses the second assay (typically the normalized one).
factors	Character vector. Column names in colData(se) to include as variance components. All specified factors must exist in colData.
pca_threshold	Numeric (0-1). Minimum cumulative proportion of variance explained by retained principal components (default 0.6).
variance_threshold	Numeric (0-1). Factors explaining less than this proportion are grouped into a "Below X%" category (default 0.01).
na_action	Character. How to handle NAs: "complete" (default) removes rows with any NA, "fill" replaces NAs with fill_value, "none" assumes no NAs (errors if present).
fill_value	Numeric. Replacement for NAs when na_action = "fill" (default -1).
verbose	Logical. Print progress messages (default TRUE).

Details

Uses lme4 directly to fit mixed models, which allows automatic detection and exclusion of interaction terms where the number of levels equals or exceeds the number of observations (e.g., Condition:Patient in paired designs with one observation per patient-condition combination).

Value

data.frame with columns:

label Factor name, interaction term, "Below X%", or "resid"

weights Weighted average proportion of variance (0-1)

Examples

```
data(nadia_dia)

# nadia_dia has no batch information, so the example builds a plausible one:
# two digestion batches crossed with the three conditions.
cov <- data.frame(Column = nadia_dia$metadata$Coding,
                  Batch = rep(c("b1", "b2"),
                             length.out = nrow(nadia_dia$metadata)),
                  stringsAsFactors = FALSE)
res <- process_proteomics(nadia_dia, covariate_df = cov, verbose = FALSE)

# Weight of each factor, and of their interaction, on the total variance
vc <- pvca_compute(res$se_proc, assay_name = "Impseqrob_min",
                  factors = c("Condition", "Batch"), verbose = FALSE)

vc
```

pvca_plot

Plot PVCA variance components

Description

Creates a bar plot showing the weighted average proportion of variance attributed to each factor, colored by category (technical, biological, interaction, residual).

Usage

```
pvca_plot(
  pvca_res,
  technical_factors = NULL,
  biological_factors = NULL,
  variance_threshold = 0.01,
  colors = NULL,
  title = NULL,
  base_size = 15
)
```

Arguments

pvca_res data.frame. Output from `pvca_compute()` with columns 'label' and 'weights'. Optionally pre-categorized with 'category'.

technical_factors	Character vector. Factor names classified as technical. Required if pvca_res lacks a 'category' column.
biological_factors	Character vector. Factor names classified as biological. Required if pvca_res lacks a 'category' column.
variance_threshold	Numeric. Used for categorization if pvca_res lacks 'category' (default 0.01).
colors	Named character vector of 4 colors for categories: "residual", "biological", "biol:techn", "technical". If NULL, defaults are used.
title	Character. Plot title (default NULL).
base_size	Numeric. Base font size for theme_classic (default 15).

Value

ggplot2 object

Examples

```
data(nadia_dia)

# nadia_dia has no batch information, so the example builds a plausible one:
# two digestion batches crossed with the three conditions.
cov <- data.frame(Column = nadia_dia$metadata$Coding,
                  Batch = rep(c("b1", "b2"),
                             length.out = nrow(nadia_dia$metadata)),
                  stringsAsFactors = FALSE)
res <- process_proteomics(nadia_dia, covariate_df = cov, verbose = FALSE)

vc <- pvca_compute(res$se_proc, assay_name = "Impseqprob_min",
                  factors = c("Condition", "Batch"), verbose = FALSE)
pvca_plot(vc, technical_factors = "Batch",
          biological_factors = "Condition")
```

quant_list_widget	<i>Full Widget for the Protein_QUANT Table with the log2 Matrix Appended</i>
-------------------	--

Description

Reactable table that reproduces the layout of the Protein-List_QUANT Excel sheet, with 2-level headers, sticky columns, a blue palette, aggregated filters and Excel export. It appends to Protein_QUANT the 16 columns of the normalized/imputed log2 matrix, matching by Protein Groups.

Usage

```
quant_list_widget(
  data = NULL,
  matrix_data = NULL,
  metadata = NULL,
  page_size = 15,
  height = 720,
  element_id = "quant_id_table",
  selection = NULL,
  searchable = TRUE
)
```

Arguments

<code>data</code>	Data frame or path to a Protein_QUANT TSV/CSV/Parquet file. Required; there is no default.
<code>matrix_data</code>	Data frame or path to a TSV/CSV/Parquet file with the normalized/imputed log2 matrix (a ProteinGroups column + _ samples).
<code>metadata</code>	Optional: metadata data frame with a Coding column that fixes the sample order.
<code>page_size</code>	Rows per page (default 15).
<code>height</code>	Table height in px (default 720).
<code>element_id</code>	Reactable element ID.
<code>selection</code>	"single" "multiple" NULL.
<code>searchable</code>	Enable the search box (default TRUE).

Value

A browsable object.

Examples

```
if (requireNamespace("jsonlite", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  # The log2 matrix: a ProteinGroups column plus one column per sample
  mat <- SummarizedExperiment::assay(res$se_proc, "Impseqprob_min")
  mat_df <- data.frame(ProteinGroups = rownames(mat), mat,
    check.names = FALSE)

  w <- quant_list_widget(nadia_dia$protein_quant, matrix_data = mat_df,
    metadata = nadia_dia$metadata)
  print(class(w))
}
```

read_nadia	<i>Read a .nadia file</i>
------------	---------------------------

Description

Loads the whole analysis: the preprocessing tables, the processing results, the intensity matrices, the parameters and the provenance. The canonical tables come out of the SQL views stored in the file, with their R classes and column order restored.

Usage

```
read_nadia(file)
```

Arguments

file	Path to a .nadia file.
------	------------------------

Details

To get objects the rest of the package can use directly, pass the result to [nadia_preprocessing\(\)](#) or [nadia_result\(\)](#).

Value

An object of class `nadia_db`: a list with `file`, `meta`, `parameters`, `calls`, `packages`, `source_files`, the canonical tables (`metadata`, `protein_id`, `protein_quant` and, when present, `DEPs_results`, `BoxPlot_Input`, `PCA_Input`, `pattern_profiler`), the matrices and the `imputed_values` mask.

See Also

[write_nadia\(\)](#), [nadia_result\(\)](#), [nadia_preprocessing\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  f <- file.path(tempdir(), "read.nadia")
  write_nadia(f, nadia_dia, res, verbose = FALSE)

  db <- read_nadia(f)
  db
  identical(db$protein_quant, nadia_dia$protein_quant)

  unlink(f)
}
```

`read_pattern_profiler_data`*Read Pattern Profiler data from a parquet file*

Description

Read Pattern Profiler data from a parquet file

Usage

```
read_pattern_profiler_data(file_path, min_membership = NULL)
```

Arguments

`file_path` Path to the parquet file
`min_membership` Optional filter by minimum membership

Value

DataFrame with the clustering data

Examples

```
if (requireNamespace("arrow", quietly = TRUE) &&  
    requireNamespace("Mfuzz", quietly = TRUE) &&  
    requireNamespace("e1071", quietly = TRUE)) {  
  data(nadia_dia)  
  res <- process_proteomics(nadia_dia, verbose = FALSE)  
  
  # Write the clustering to parquet, then read it back  
  f <- file.path(tempdir(), "pattern_profiler.parquet")  
  pattern_profiler_analysis(res$se_proc, res$DEPs_results,  
                           assay_name = "Impseqrob_min",  
                           auto_select_c = FALSE, c = 3,  
                           output_file = f, verbose = FALSE)  
  
  pp_data <- read_pattern_profiler_data(f, min_membership = 0.5)  
  unlink(f)  
  print(head(pp_data))  
}
```

results_list_reactable

Interactive Reactable Table for Differential Expression Results

Description

Builds a professionally formatted reactable table to display protein differential expression results. Works both in Shiny (it returns a reactable object) and standalone via results_list_widget().

Usage

```
results_list_reactable(
  data,
  protein_quant = NULL,
  comparisons = NULL,
  ain = NULL,
  alpha = 0.05,
  lfc_thr = 0,
  page_size = 15,
  height = 720,
  show_missing = TRUE,
  element_id = NULL,
  selection = NULL,
  searchable = TRUE
)
```

Arguments

data	Data frame or path to a TSV/CSV/Parquet file with DE results. Required columns: Protein.IDs, Gene.Names, logFC, P.Value, adj.P.Val, Change, Comparison. Optional: Assay, and the four missingness columns MissGlobal, MissComp, MissCND1 and MissCND2, which are shown only if all four are present
protein_quant	Data frame (preprocessing\$protein_quant) or path to a Protein_QUANT_*.tsv file. If not NULL, the Description and Quant_Pepts columns are added.
comparisons	Vector of comparisons to include (NULL = all of them)
ain	Vector of assays to keep (NULL = all of them)
alpha	Significance threshold used to highlight the FDR (default: 0.05)
lfc_thr	log2 fold-change threshold (default: 0, reserved for future use)
page_size	Rows per page (default: 15)
height	Table height in pixels (default: 720)
show_missing	Show the missingness percentage columns (default: TRUE)
element_id	Element ID for the Reactable JS API (default: NULL)
selection	Selection type: "single", "multiple", or NULL (default: NULL)
searchable	Enable the internal search box (default: TRUE)

Value

A reactable object

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# protein_quant adds the Description and Quant_Pepts columns
tbl <- results_list_reactable(
  res$DEPs_results,
  protein_quant = nadia_dia$protein_quant,
  comparisons   = "B-A"
)
class(tbl)
```

results_list_widget *Full Widget with Filters, Search, Export and CSS*

Description

Wraps results_list_reactable() with interactive crosstalk filters (Comparison, Change, Method), a search box, an export-to-Excel button and embedded CSS. The result is browsable: printing it at the console opens it automatically in the RStudio Viewer or in the browser.

Usage

```
results_list_widget(
  data,
  protein_quant = NULL,
  comparisons = NULL,
  ain = NULL,
  alpha = 0.05,
  lfc_thr = 0,
  page_size = 15,
  height = 720,
  show_missing = TRUE,
  element_id = "deps_table",
  selection = NULL,
  searchable = TRUE
)
```

Arguments

data	Data frame or path to a TSV/CSV/Parquet file with DE results. Required columns: Protein.IDs, Gene.Names, logFC, P.Value, adj.P.Val, Change, Comparison. Optional: Assay, and the four missingness columns MissGlobal, MissComp, MissCND1 and MissCND2, which are shown only if all four are present
------	---

protein_quant	Data frame (preprocessing\$protein_quant) or path to a Protein_QUANT_*.tsv file. If not NULL, the Description and Quant_Pepts columns are added.
comparisons	Vector of comparisons to include (NULL = all of them)
ain	Vector of assays to keep (NULL = all of them)
alpha	Significance threshold used to highlight the FDR (default: 0.05)
lfc_thr	log2 fold-change threshold (default: 0, reserved for future use)
page_size	Rows per page (default: 15)
height	Table height in pixels (default: 720)
show_missing	Show the missingness percentage columns (default: TRUE)
element_id	Element ID (default: "deps_table")
selection	Selection type: "single", "multiple", or NULL (default: NULL)
searchable	Enable the internal search box (default: TRUE)

Value

A browsable htmltools object

Examples

```
if (requireNamespace("crosstalk", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  # Printing the widget opens it in the Viewer / browser
  w <- results_list_widget(res$DEPs_results,
    protein_quant = nadia_dia$protein_quant)
  print(class(w))
}
```

run_mfuzz_clustering *Run Mfuzz clustering*

Description

Run Mfuzz clustering

Usage

```
run_mfuzz_clustering(eset_std, c, m, seed = 42)
```

Arguments

eset_std	Standardised ExpressionSet
c	Number of clusters
m	Fuzziness parameter
seed	Seed for reproducibility

Value

Mfuzz clustering object

select_optimal_clusters

Select the optimal number of clusters

Description

Select the optimal number of clusters

Usage

```
select_optimal_clusters(  
  eset_std,  
  c_range = 2:10,  
  m = NULL,  
  method = c("xb", "consensus", "elbow"),  
  seeds = c(42, 123, 456),  
  verbose = TRUE  
)
```

Arguments

eset_std	Standardised ExpressionSet
c_range	Range of cluster numbers to evaluate
m	Fuzziness parameter
method	Method: "xb", "consensus", "elbow"
seeds	Seeds passed on to evaluate_cluster_range()
verbose	Show progress

Value

List with optimal_c, metrics, m

standardize_eset	<i>Standardise an ExpressionSet (row-wise z-score)</i>
------------------	--

Description

Standardise an ExpressionSet (row-wise z-score)

Usage

```
standardize_eset(eset, verbose = TRUE)
```

Arguments

- eset ExpressionSet object
- verbose Print progress messages (default: TRUE)

Value

Standardised ExpressionSet

summarize_pattern_profiler	<i>Summary of the Pattern Profiler data</i>
----------------------------	---

Description

Summary of the Pattern Profiler data

Usage

```
summarize_pattern_profiler(data)
```

Arguments

- data Pattern Profiler DataFrame

Value

List with summary statistics

Examples

```
if (requireNamespace("Mfuzz", quietly = TRUE) &&
    requireNamespace("e1071", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)
  pp <- pattern_profiler_analysis(res$se_proc, res$DEPs_results,
                                assay_name = "Impseqrob_min",
                                auto_select_c = FALSE, c = 3,
                                verbose = FALSE)

  info <- summarize_pattern_profiler(pp$long_output)
  print(info$cluster_summary)
}
```

summary_list_widget *Interactive Reactable Table for the Sample Metadata*

Description

Builds a reactable table that reproduces the layout of the metadata Excel sheet (FileName, Condition, Replicate, Coding, # Unique PSMs, # Unique Peptides, # Protein Groups). Black header, rows tinted by condition in light shades and Condition/Coding cells with a strong colour chip (the same scheme used by protein_list_widget() / quant_list_widget()).

Usage

```
summary_list_widget(
  data = NULL,
  page_size = 16,
  height = 540,
  element_id = "summary_table",
  selection = NULL,
  searchable = FALSE
)
```

Arguments

data	Data frame or path to a metadata TSV/CSV file. Required; there is no default. It must have the columns R.FileName, R.Condition, R.Replicate, Coding, R.PrecursorsIdentified, R.StrippedSequencesIdentified, R.ProteinGroupsIdentified.
page_size	Page size (default 16, i.e. all the rows).
height	Height in px of the table container (default 540).
element_id	Widget id in the DOM (default "summary_table").
selection	Selection type ("multiple" or NULL).
searchable	Enable the global search box (default FALSE).

Value

An htmltools object (browsable tagList) ready to be rendered.

Examples

```
if (requireNamespace("jsonlite", quietly = TRUE)) {
  data(nadia_dia)

  # LFQ/TMT metadata lack the identification counts; the widget then shows
  # only FileName/Condition/Replicate/Coding
  w <- summary_list_widget(nadia_dia$metadata)
  print(class(w))
}
```

volcano_highchart_list

Interactive Volcano Plot with Highcharter

Description

Interactive Volcano Plot with Highcharter

Usage

```
volcano_highchart_list(
  de_res,
  ain = NULL,
  comparisons = NULL,
  lfc_thr = 0,
  alpha = 0.05,
  p_col = "adj.P.Val",
  point_size = 4,
  colors = NULL,
  palette = NULL,
  show_top_genes = 0,
  highlight_genes = NULL,
  title = NULL
)
```

Arguments

de_res	Data frame with differential expression results
ain	Vector of assays to filter on (optional)
comparisons	Vector of comparisons to include (optional)
lfc_thr	log2 fold-change threshold (default: 0)

alpha	Significance threshold (default: 0.05)
p_col	Column of p-values to use
point_size	Point size (default: 4)
colors	List of colours for "up", "down", "ns" (ignored when palette is used)
palette	Name of a palleteer palette (e.g. "ggsci::default_jco"). Uses 3 colours: up, down, ns
show_top_genes	Number of top genes to label by significance (default: 0)
highlight_genes	Vector of gene names to highlight manually (default: NULL)
title	Custom chart title (default: NULL, uses "Comparison (Assay)"). Use {comparison} as a placeholder (e.g. "Volcano Plot: {comparison}" -> "Volcano Plot: B-A")

Value

List of highchart objects

Examples

```
data(nadia_dia)
res <- process_proteomics(nadia_dia, verbose = FALSE)

# One volcano per comparison, labelling the 10 most significant proteins
hc_list <- volcano_highchart_list(
  res$DEPs_results,
  comparisons = c("B-A", "D-A"),
  lfc_thr = 1,
  alpha = 0.05,
  show_top_genes = 10
)
names(hc_list)
```

wrap_heatmap_with_title

Wrap a Heatmap Together With Its Title

Description

Wrap a Heatmap Together With Its Title

Usage

```
wrap_heatmap_with_title(hm, title, title_size = 14, title_face = "bold")
```

Arguments

hm	tidyHeatmap object (InputHeatmap)
title	Heatmap title
title_size	Font size of the title
title_face	Font face of the title

Value

proteomics_heatmap object (S3 list)

write_nadia	<i>Write a complete analysis to a single .nadia file</i>
-------------	--

Description

Stores the preprocessing tables, the processing results and, optionally, the Pattern Profiler output in one DuckDB database, together with the parameters and the provenance of the run. Nothing is duplicated: everything that can be derived is left out and rebuilt on reading, through SQL views stored in the file itself.

Usage

```
write_nadia(
  file,
  preprocessing,
  result = NULL,
  pattern_profiler = NULL,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

file	Path to write to. The .nadia extension is a convention; the file is a DuckDB database whatever it is called.
preprocessing	A proteomics_data object, from any of the preprocess_*() functions. Required: it is the base of the file.
result	Object from process_proteomics() , or NULL to store only the preprocessing.
pattern_profiler	Object from pattern_profiler_analysis() , or NULL. Can also be added later with nadia_add_pattern_profiler() .
overwrite	Replace file if it already exists.
verbose	Print progress.

Details

Use `read_nadia()` to read it back, and `nadia_result()` to get an object the rest of the package can plot directly.

Value

The path, invisibly.

What goes in

Twelve tables. `samples` and `proteins` are the dimensions; `protein_sample_metrics`, `intensities`, `imputed_values` and `de_results` the facts; `pp_membership` and `pp_profile` the optional clustering; and `nadia_meta`, `nadia_packages`, `nadia_source_files`, `nadia_calls`, `parameters` and `nadia_schema` the provenance.

Three things are deliberately *not* stored, because they are exactly derivable: the log2 assay (log2 of the raw one), the `PCA_Input` table (the imputed assay joined to the DE results) and the observed cells of the imputed assay (identical to the normalized ones). The last of these is checked rather than assumed, since a model-based method such as `limpa` can revise observed values; when it does, the full assay is stored instead.

Where not to put it

Not inside a synchronised folder such as iCloud Drive, Dropbox or OneDrive while you are working. DuckDB keeps the file open with locks and partial writes, and a synchronising client can corrupt it. Write it locally and copy it afterwards.

See Also

`read_nadia()`, `nadia_result()`, `nadia_add_pattern_profiler()`

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  data(nadia_dia)
  res <- process_proteomics(nadia_dia, verbose = FALSE)

  f <- file.path(tempdir(), "example.nadia")
  write_nadia(f, nadia_dia, res, verbose = FALSE)

  db <- read_nadia(f)
  db

  unlink(f)
}
```

Index

* datasets

nadia_dia, [75](#)

base::system.file(), [80](#)

batch_correct_proteomics, [5](#)

benchmark_aucBars_gg, [7](#)

benchmark_confusion_gg, [8](#)

benchmark_confusion_overall_gg, [9](#)

benchmark_heatmap_gg, [10](#)

benchmark_metrics_bars_gg, [11](#)

benchmark_roc_gg, [12](#)

benchmark_signif_bars_gg, [13](#)

benchmark_volcano_hc, [14](#)

benchmark_volcano_hc_list, [15](#)

benchmarking_multiple, [16](#)

benchmarking_proteomics, [19](#)

bm_compute_extended_ranking, [21](#)

bm_compute_ranking, [22](#)

bm_compute_ranking_by_comparison, [23](#)

bm_plot_confusion_stacked, [24](#)

bm_plot_extended_ranking_bars, [25](#)

bm_plot_extended_ranking_heatmap, [26](#)

bm_plot_metrics_comparison, [27](#)

bm_plot_metrics_heatmap, [28](#)

bm_plot_ranking_bars, [29](#)

bm_plot_ranking_bars_by_comp, [29](#)

bm_plot_ranking_heatmap, [30](#)

bm_plot_ranking_heatmap_by_comp, [31](#)

bm_plot_roc, [32](#)

boxplot_highchart_list, [33](#)

boxplot_highchart_list(), [131](#)

build_clustering_matrix, [34](#)

build_long_output, [35](#)

build_pca_scores, [36](#)

cluster_centroids_highchart, [37](#)

cluster_profile_highchart, [38](#)

cluster_profile_highchart_list, [40](#)

cluster_profile_highchart_list(), [77](#)

compute_benchmark_metrics, [41](#)

compute_confidence_ellipse, [42](#)

compute_dispersion_metrics, [43](#)

compute_hulls, [44](#)

compute_opdea_metrics, [44](#)

configure_cluster_palette, [46](#)

create_expression_set, [46](#)

de_analysis_proteomics, [47](#)

deps_with_clusters, [48](#)

deps_with_clusters(), [73](#), [74](#)

detect_condition_columns, [50](#)

evaluate_cluster_range, [51](#)

extract_se_data, [51](#)

filter_significant_features, [52](#)

get_annotation_palette, [53](#)

get_heatmap_palette, [53](#)

im_compute_metrics, [54](#)

im_plot_acc_oi, [55](#)

im_plot_metrics, [56](#)

im_plot_nrmse, [56](#)

im_plot_pss, [57](#)

im_plot_ranking, [58](#)

im_plot_sor, [58](#)

im_prepare_se, [59](#)

import_benchmark_metrics, [60](#)

import_classified_results, [61](#)

import_confusion_results, [62](#)

import_imp_matrices, [63](#)

import_norm_matrices, [64](#)

import_opdea_results, [66](#)

imputation_metrics, [67](#)

impute_proteomics, [68](#)

merge_significance_info, [71](#)

nadia_add_pattern_profiler, [71](#)

- `nadia_add_pattern_profiler()`, 74, 77, 158, 159
- `nadia_connect`, 72
- `nadia_deps_with_clusters`, 73
- `nadia_deps_with_clusters()`, 50
- `nadia_dia`, 75
- `nadia_export_parquet`, 76
- `nadia_pattern_profiler`, 77
- `nadia_preprocessing`, 78
- `nadia_preprocessing()`, 79, 148
- `nadia_result`, 79
- `nadia_result()`, 78, 148, 159
- `nadia_skill_path`, 80
- `nadia_tables`, 81
- `nadia_tables()`, 73
- `nm_compute_metrics`, 81
- `nm_plot_boxplot`, 83
- `nm_plot_correlation`, 84
- `nm_plot_density`, 85
- `nm_plot_final_ranking`, 85
- `nm_plot_mds`, 86
- `nm_plot_mds1_ranking`, 87
- `nm_plot_metrics`, 88
- `nm_plot_pc1_ranking`, 89
- `nm_plot_pca`, 90
- `nm_plot_pcv`, 91
- `nm_plot_pev`, 92
- `nm_plot_pmad`, 93
- `nm_plot_qq`, 94
- `nm_plot_scatter`, 95
- `nm_prepare_se`, 96
- `nm_rank_cor`, 97
- `nm_rank_final`, 98
- `nm_rank_mds1`, 99
- `nm_rank_pc1`, 100
- `nm_rank_pcv`, 101
- `nm_rank_pev`, 102
- `nm_rank_pmad`, 103
- `nm_run_normalizations`, 104
- `normalization_metrics`, 105
- `normalize_proteomics`, 107
- `pattern_profiler_analysis`, 108
- `pattern_profiler_analysis()`, 49, 50, 72, 75, 158
- `pca_covariates_plot`, 110
- `pca_highchart`, 112
- `pca_highchart_list`, 114
- `pca_highchart_list()`, 131
- `prepare_heatmap_data`, 115
- `preprocess_diann`, 116
- `preprocess_diann()`, 107, 118, 123, 129
- `preprocess_lfq`, 118
- `preprocess_lfq()`, 107, 124, 129
- `preprocess_spectronaut`, 120
- `preprocess_spectronaut()`, 75, 107, 127, 129
- `preprocess_tmt`, 121
- `preprocess_tmt()`, 107, 118, 127, 129
- `print.diann_data`, 123
- `print.lfq_data`, 124
- `print.nadia_db`, 125
- `print.proteomics_heatmap`, 125
- `print.proteomics_result`, 126
- `print.spectronaut_data`, 127
- `print.tmt_data`, 127
- `process_proteomics`, 128
- `process_proteomics()`, 49, 75, 78, 79, 116, 117, 119, 122, 158
- `protein_list_reactable`, 132
- `protein_list_widget`, 133
- `proteomics_heatmap`, 134
- `proteomics_heatmap()`, 125, 131
- `proteomics_heatmap_list`, 138
- `proteomics_heatmap_list()`, 131
- `pvca_analysis`, 142
- `pvca_compute`, 144
- `pvca_plot`, 145
- `quant_list_widget`, 146
- `read_nadia`, 148
- `read_nadia()`, 72–74, 76–79, 81, 159
- `read_pattern_profiler_data`, 149
- `results_list_reactable`, 150
- `results_list_widget`, 151
- `run_mfuzz_clustering`, 152
- `select_optimal_clusters`, 153
- `standardize_eset`, 154
- `summarize_pattern_profiler`, 154
- `summary_list_widget`, 155
- `volcano_highchart_list`, 156
- `wrap_heatmap_with_title`, 157
- `write_nadia`, 158
- `write_nadia()`, 72, 77, 118, 119, 121, 123, 148