Package 'proBatch'

May 8, 2026

Type Package

Title Tools for Diagnostics and Corrections of Batch Effects in Proteomics

Version 2.0.0

Description These tools facilitate batch effects analysis and correction in high-throughput experiments. It was developed primarily for mass-spectrometry proteomics (DIA/SWATH),
but could also be applicable to most omic data with minor adaptations. The package contains functions
for diagnostics (proteome/genome-wide and feature-level), correction (normalization and batch effects correction) and quality control. Non-linear fitting based approaches were also included to deal with complex, mass spectrometry-specific signal drifts.

biocViews BatchEffect, Normalization, Preprocessing, Software, MassSpectrometry, Proteomics, QualityControl, Visualization

License GPL-3

URL https://github.com/Freddsle/proBatch

BugReports https://github.com/Freddsle/proBatch/issues

Depends R (>= 4.5.0)

Encoding UTF-8

LazyData false

Imports Biobase, QFeatures, SummarizedExperiment, S4Vectors, corrplot, dplyr, data.table, ggfortify, ggplot2, gridExtra, grDevices, lazyeval, lubridate, limma, magrittr, matrixStats, methods, pheatmap, preprocessCore, purrr, pvca, RColorBrewer, reshape2, rlang, scales, stats, sva, tidyr, tibble, tools, utils, viridis, wesanderson, WGCNA

Suggests BiocStyle, cowplot, ggplotify, knitr, rmarkdown, devtools, gtable, roxygen2, testthat (>= 3.0.0), spelling, HDF5Array

VignetteBuilder knitr

RoxygenNote 7.3.3

Language en-US

Config/testthat/edition 3

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

git_branch RELEASE_3_23

git_last_commit 06dc661

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-05-07

Author Jelena Cuklina [aut],
Chloe H. Lee [aut],
Patrick Pedrioli [aut],
Olga Zolotareva [aut],
Yuliya Burankova [cre]

Maintainer Yuliya Burankova <yuliya.burankova@uni-hamburg.de>

Contents

`calculate_feature_CV` *Calculate CV distribution for each feature*

Description

Calculate CV distribution for each feature

Usage

```
calculate_feature_CV(
  df_long,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  batch_col = NULL,
  biospecimen_id_col = NULL,
  unlog = TRUE,
  log_base = 2,
  offset = 0
)
```

Arguments

`df_long` data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`measure_col` if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

`batch_col` column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

biospecimen_id_col
: column in sample_annotation that defines a unique bio ID, which is usually a combination of conditions or groups. Tip: if such ID is absent, but can be defined from several columns, create new biospecimen_id column

unlog
: (logical) whether to reverse log transformation of the original data

log_base
: base of the logarithm for transformation

offset
: small positive number to prevent 0 conversion to -Inf

Value

data frame with Total CV for each feature & (optionally) per-batch CV

Examples

```
data(list = c("example_sample_annotation", "example_proteome"), package = "proBatch")
CV_df <- calculate_feature_CV(example_proteome,
    sample_annotation = example_sample_annotation,
    measure_col = "Intensity",
    batch_col = "MS_batch"
)
```

---

calculate_peptide_corr_distr

Calculate peptide correlation between and within peptides of one protein

---

Description

Calculate peptide correlation between and within peptides of one protein

Usage

```
calculate_peptide_corr_distr(
  data_matrix,
  peptide_annotation,
  protein_col = "ProteinName",
  feature_id_col = "peptide_group_label"
)
```

Arguments

data_matrix
: features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use help("example_proteome_matrix"))

peptide_annotation
: long format data frame with peptide ID and their corresponding protein and/or gene annotations. See help("example_peptide_annotation").

protein_col
: column where protein names are specified

feature_id_col
: name of the column with feature/gene/peptide/protein ID used in the long format representation df_long. In the wide formatted representation data_matrix this corresponds to the row names.

Value

dataframe with peptide correlation coefficients that are suggested to use for plotting in plot_peptide_corr_distribution as plot_param:

Examples

```
data(list = c("example_peptide_annotation", "example_proteome_matrix"), package = "proBatch")
selected_genes <- c("BOVINE_A1ag", "BOVINE_FetuinB", "Cyfip1")
gene_filter <- example_peptide_annotation$Gene %in% selected_genes
peptides_ann <- example_peptide_annotation$peptide_group_label
selected_peptides <- peptides_ann[gene_filter]
matrix_test <- example_proteome_matrix[selected_peptides, ]
pep_annotation_sel <- example_peptide_annotation[gene_filter, ]
corr_distribution <- calculate_peptide_corr_distr(matrix_test,
    pep_annotation_sel,
    protein_col = "Gene"
)
```

---

calculate_PVCA   *Calculate variance distribution by variable*

Description

Calculate variance distribution by variable

Usage

```
## Default S3 method:
calculate_PVCA(
  data_matrix,
  sample_annotation,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  factors_for_PVCA = c("MS_batch", "digestion_batch", "Diet", "Sex", "Strain"),
  pca_threshold = 0.6,
  variance_threshold = 0.01,
  fill_the_missing = -1,
  ...
)

## S3 method for class 'ProBatchFeatures'
```

```
calculate_PVCA(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  ...
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`factors_for_PVCA` vector of factors from `sample_annotation`, that are used in PVCA analysis

`pca_threshold` the percentile value of the minimum amount of the variabilities that the selected principal components need to explain

`variance_threshold` the percentile value of weight each of the factors needs to explain (the rest will be lumped together)

`fill_the_missing` numeric value determining how missing values should be substituted. If `NULL`, features with missing values are excluded.

`...` Additional arguments forwarded between methods.

`pbf_name` Assay name(s) used when ‘data_matrix‘ is a ‘ProBatchFeatures‘.

Value

data frame of weights of Principal Variance Components

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
matrix_test <- na.omit(example_proteome_matrix)[1:50, ]
```

```
pvca_df <- calculate_PVCA(matrix_test, example_sample_annotation,
    factors_for_PVCA = c("MS_batch", "digestion_batch", "Diet", "Sex", "Strain"),
    pca_threshold = .6, variance_threshold = .01, fill_the_missing = -1
)
```

`calculate_sample_corr_distr`

Calculates correlation for all pairs of the samples in data matrix, labels as replicated/same_batch/unrelated in output columns (see "Value").

Description

Calculates correlation for all pairs of the samples in data matrix, labels as replicated/same_batch/unrelated in output columns (see "Value").

Usage

```
calculate_sample_corr_distr(
  data_matrix,
  sample_annotation,
  repeated_samples = NULL,
  biospecimen_id_col = "EarTag",
  sample_id_col = "FullRunName",
  batch_col = "MS_batch"
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`repeated_samples` vector of sample IDs to evaluate, if `NULL`, all samples are taken into account for plotting

`biospecimen_id_col` column in `sample_annotation` that defines a unique bio ID, which is usually a combination of conditions or groups. Tip: if such ID is absent, but can be defined from several columns, create new `biospecimen_id` column sample_id_col — name of the column in sample_annotation table, where the filenames (colnames of the data_matrix are found).

batch_col — column in sample_annotation that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

Value

dataframe with the following columns, that are suggested to use for plotting in plot_sample_corr_distribution as plot_param:

1. replicate
2. batch_the_same
3. batch_replicate
4. batches

other columns are:

1. sample_id_1 & sample_id_2, both generated from sample_id_col variable
2. correlation - correlation of two corresponding samples
3. batch_1 & batch_2 or analogous, created the same as sample_id_1

Examples

```
data(list = c("example_sample_annotation", "example_proteome_matrix"), package = "proBatch")
corr_distribution <- calculate_sample_corr_distr(
    data_matrix = example_proteome_matrix,
    sample_annotation = example_sample_annotation,
    batch_col = "MS_batch", biospecimen_id_col = "EarTag"
)
```

---

check_sample_consistency — *Check if sample annotation is consistent with data matrix and join the two*

Description

Check if sample annotation is consistent with data matrix and join the two

Usage

```
check_sample_consistency(
  sample_annotation,
  sample_id_col,
  df_long,
  batch_col = NULL,
```

```
  order_col = NULL,
  facet_col = NULL,
  merge = TRUE
)
```

Arguments

`sample_annotation`
: data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`sample_id_col`
: name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`df_long`
: data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.

`batch_col`
: column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

`order_col`
: column in `sample_annotation` that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

`facet_col`
: column in `sample_annotation` with a batch factor to separate plots into facets; usually 2nd to `batch_col`. Most meaningful for multi-instrument MS experiments (where each instrument has its own order-associated effects (see `order_col`) or simultaneous examination of two batch factors (e.g. preparation day and measurement day). For single-instrument case should be set to 'NULL'

`merge`
: (logical) whether to merge `df_long` with `sample_annotation` or not

Value

`df_long` format data frame, merged with sample_annotation using inner_join (samples represented in both)

Examples

```
# Load necessary datasets
data(list = c("example_proteome", "example_sample_annotation"), package = "proBatch")

df_test <- check_sample_consistency(
    sample_annotation = example_sample_annotation,
    df_long = example_proteome, sample_id_col = "FullRunName",
    batch_col = NULL, order_col = NULL, facet_col = NULL
```

```
)
```

`color_list_to_df` *Color list to data frame*

Description

Turn color list to df (to use in the hierarchical clustering)

Usage

```
color_list_to_df(color_list, sample_annotation, sample_id_col = "FullRunName")
```

Arguments

`color_list`	list of colors
`sample_annotation`	factor-based configuration of the sample annotation

Value

a data frame representation of the input color list

`convert_annotation_classes` *Convert factor and numeric columns*

Description

Convert specified factor columns to factor type and numeric columns to numeric.

Usage

```
convert_annotation_classes(df, factor_columns, numeric_columns)
```

Arguments

`df`	data frame with sample annotations.
`factor_columns`	character vector of factor columns.
`numeric_columns`	character vector of numeric columns.

Value

data frame with converted columns.

correct_batch_effects *Batch correction of normalized data*

Description

Batch correction of normalized data. Batch correction brings each feature in each batch to the comparable shape. Currently the following batch correction functions are implemented:

1. Per-feature median centering: `center_feature_batch_medians_df()`. Median centering of the features (per batch median).
2. correction with ComBat: `correct_with_ComBat_df()`. Adjusts for discrete batch effects using ComBat. ComBat, described in Johnson et al. 2007. It uses either parametric or non-parametric empirical Bayes frameworks for adjusting data for batch effects. Users are returned an expression matrix that has been corrected for batch effects. The input data are assumed to be free of missing values and normalized before batch effect removal. Please note that missing values are common in proteomics, which is why in some cases corrections like `center_peptide_batch_medians_df` are more appropriate.
3. Continuous drift correction: `adjust_batch_trend_df()`. Adjust batch signal trend with the custom (continuous) fit. Should be followed by discrete corrections, e.g. `center_feature_batch_medians_df()` or `correct_with_ComBat_df()`.

Alternatively, one can call the correction function with `correct_batch_effects_df()` wrapper. Batch correction method allows correction of continuous signal drift within batch (if required) and adjustment for discrete difference across batches.

Usage

```
center_feature_batch_medians_df(
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  keep_all = "default",
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = NULL
)

center_feature_batch_medians_dm(
  data_matrix,
  sample_annotation,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity"
```

```
)

center_feature_batch_means_df(
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  keep_all = "default",
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = NULL
)

center_feature_batch_means_dm(
  data_matrix,
  sample_annotation,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity"
)

adjust_batch_trend_df(
  df_long,
  sample_annotation = NULL,
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  order_col = "order",
  keep_all = "default",
  fit_func = "loess_regression",
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = NULL,
  min_measurements = 8,
  ...
)

adjust_batch_trend_dm(
  data_matrix,
  sample_annotation,
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
```

```
  order_col = "order",
  fit_func = "loess_regression",
  return_fit_df = TRUE,
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = NULL,
  min_measurements = 8,
  ...
)

correct_with_ComBat_df(
  df_long,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  par.prior = TRUE,
  fill_the_missing = NULL,
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = NULL,
  keep_all = "default"
)

correct_with_ComBat_dm(
  data_matrix,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  par.prior = TRUE,
  fill_the_missing = NULL
)

correct_batch_effects_df(
  df_long,
  sample_annotation,
  continuous_func = NULL,
  discrete_func = c("MedianCentering", "MeanCentering", "ComBat"),
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  order_col = "order",
  keep_all = "default",
  no_fit_imputed = TRUE,
```

```
  qual_col = NULL,
  qual_value = NULL,
  fill_the_missing = NULL,
  min_measurements = 8,
  ...
)

correct_batch_effects_dm(
  data_matrix,
  sample_annotation,
  continuous_func = NULL,
  discrete_func = c("MedianCentering", "ComBat"),
  batch_col = "MS_batch",
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  order_col = "order",
  min_measurements = 8,
  no_fit_imputed = TRUE,
  fill_the_missing = NULL,
  ...
)
```

Arguments

- `df_long` — data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.
- `sample_annotation` — data frame with:
  1. `sample_id_col` (this can be repeated as row names)
  2. biological covariates
  3. technical covariates (batches etc)

  . See `help("example_sample_annotation")`
- `sample_id_col` — name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).
- `batch_col` — column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).
- `feature_id_col` — name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.
- `measure_col` — if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.
- `keep_all` — when transforming the data (normalize, correct) - acceptable values: all/default/minimal (which set of columns be kept).

no_fit_imputed (logical) whether to use imputed (requant) values, as flagged in qual_col by qual_value for data transformation

qual_col column to color point by certain value denoted by qual_value. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to m_score.

qual_value value in qual_col to color. For OpenSWATH data, this argument value has to be set to 2 (this is an m_score value for imputed values (requant values).

data_matrix features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use help("example_proteome_matrix"))

order_col column in sample_annotation that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

fit_func function to fit the (non)-linear trend

min_measurements
the number of samples in a batch required for curve fitting.

... other parameters, usually of adjust_batch_trend, and fit_func.

return_fit_df (logical) whether to return the fit_df from adjust_batch_trend_dm or only the data matrix

par.prior use parametrical or non-parametrical prior

fill_the_missing
numeric value used to impute missing measurements before correction. If NULL (default) missing values are left untouched, if FALSE rows with missing values are removed prior to correction.

continuous_func
function to use for the fit (currently only loess_regression available); if order-associated fix is not required, should be NULL.

discrete_func function to use for adjustment of discrete batch effects (MedianCentering or ComBat).

Value

the data in the same format as input (data_matrix or df_long). For df_long the data frame stores the original values of measure_col in another column called "preBatchCorr_[measure_col]", and the normalized values in measure_col column.

The function adjust_batch_trend_dm(), if return_fit_df is TRUE returns list of two items:

1. data_matrix
2. fit_df, used to examine the fitting curves

See Also

fit_nonlinear

fit_nonlinear, plot_with_fitting_curve

fit_nonlinear, plot_with_fitting_curve

Examples

```
data(
    list = c("example_sample_annotation", "example_proteome"),
    package = "proBatch"
)
median_centered_df <- center_feature_batch_medians_df(
    example_proteome, example_sample_annotation
)

combat_corrected_df <- correct_with_ComBat_df(
    example_proteome,
    example_sample_annotation
)

# Adjust the MS signal drift:
test_peptides <- unique(example_proteome$peptide_group_label)[1:3]
test_peptide_filter <- example_proteome$peptide_group_label %in% test_peptides
test_proteome <- example_proteome[test_peptide_filter, ]
adjusted_df <- adjust_batch_trend_df(test_proteome,
    example_sample_annotation,
    span = 0.7,
    min_measurements = 8
)
plot_fit <- plot_with_fitting_curve(
    unique(adjusted_df$peptide_group_label),
    df_long = adjusted_df,
    measure_col = "preTrendFit_Intensity",
    fit_df = adjusted_df,
    sample_annotation = example_sample_annotation
)

# Correct the data in one go:
batch_corrected_matrix <- correct_batch_effects_df(example_proteome,
    example_sample_annotation,
    continuous_func = "loess_regression",
    discrete_func = "MedianCentering",
    batch_col = "MS_batch",
    span = 0.7, min_measurements = 8
)
```

---

correct_with_removeBatchEffect_dm

Batch effect correction with removeBatchEffect from limma

Description

Batch effect correction with removeBatchEffect.

Usage

```
correct_with_removeBatchEffect_dm(
  data_matrix,
  sample_annotation,
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  covariates_cols = NULL,
  fill_the_missing = NULL,
  ...
)
```

Arguments

- `data_matrix` data matrix with features in rows and samples in columns
- `sample_annotation` data frame with sample annotations
- `feature_id_col` column name in `data_matrix` with feature IDs
- `measure_col` column name in `data_matrix` with measured values
- `sample_id_col` column name in `sample_annotation` with sample IDs
- `batch_col` column name in `sample_annotation` with batch IDs
- `covariates_cols` vector of column names in `sample_annotation` with covariates to include in the model
- `fill_the_missing` numeric value used to impute missing measurements before correction. If `FALSE` rows with missing values are removed.
- `...` other parameters to pass to `removeBatchEffect`

Value

data matrix with batch effects removed

See Also

`removeBatchEffect`

Examples

```
data(
    list = c("example_sample_annotation", "example_proteome_matrix"),
    package = "proBatch"
)
example_proteome_small <- example_proteome_matrix[1:100, ]
batch_corrected_matrix <- correct_with_removeBatchEffect_dm(
    example_proteome_small,
```

```
    example_sample_annotation,
    batch_col = "MS_batch",
    covariates_cols = c("Diet", "Sex")
)
```

---

create_peptide_annotation
Prepare peptide annotation from long format data frame

Description

Create light-weight peptide annotation data frame for selection of illustrative proteins

Usage

```
create_peptide_annotation(
  df_long,
  feature_id_col = "peptide_group_label",
  protein_col = c("ProteinName", "Gene")
)
```

Arguments

df_long	data frame where each row is a single feature in a single sample. It minimally has a sample_id_col, a feature_id_col and a measure_col, but usually also an m_score (in OpenSWATH output result file). See help("example_proteome") for more details.
feature_id_col	name of the column with feature/gene/peptide/protein ID used in the long format representation df_long. In the wide formatted representation data_matrix this corresponds to the row names.
protein_col	column where protein names are specified

Value

data frame containing peptide annotations

See Also

plot_peptides_of_one_protein, plot_protein_corrplot

Examples

```
data("example_proteome", package = "proBatch")
generated_peptide_annotation <- create_peptide_annotation(
    example_proteome,
    feature_id_col = "peptide_group_label",
    protein_col = c("Protein")
)
```

`dates_to_posix` *Convert date/time to POSIXct*

Description

convert date/time column of sample_annotation to POSIX format required to keep number-like behavior

Usage

```
dates_to_posix(
  sample_annotation,
  time_column = c("RunDate", "RunTime"),
  new_time_column = "DateTime",
  dateTimeFormat = c("%b_%d", "%H:%M:%S"),
  tz = "GMT",
  locale = "en_US.UTF-8"
)
```

Arguments

`sample_annotation`
: data frame with:
  1. `sample_id_col` (this can be repeated as row names)
  2. biological covariates
  3. technical covariates (batches etc)

  . See `help("example_sample_annotation")`

`time_column`
: name of the column(s) where run date & time are specified. These will be used to determine the run order

`new_time_column`
: name of the new column that will contain the converted date/time value

`dateTimeFormat`
: POSIX format of the date and time. See `as.POSIXct` from base R for details

`tz`
: for time zone, 'GMT' by default

`locale`
: for locale, 'en_US.UTF-8' by default

Value

sample annotation file with a new column `new_time_column` with POSIX-formatted date

Examples

```
data("example_sample_annotation", package = "proBatch")
date_to_posix <- dates_to_posix(example_sample_annotation,
    time_column = c("RunDate", "RunTime"),
    new_time_column = "DateTime_new",
    dateTimeFormat = c("%b_%d", "%H:%M:%S")
```

```
)
```

date_to_sample_order *Convert date/time to POSIXct and rank samples by it*

Description

Converts date/time columns for sample_annotation to POSIXct format and calculates sample run rank in order column

Usage

```
date_to_sample_order(
  sample_annotation,
  time_column = c("RunDate", "RunTime"),
  new_time_column = "DateTime",
  dateTimeFormat = c("%b_%d", "%H:%M:%S"),
  new_order_col = "order",
  instrument_col = "instrument"
)
```

Arguments

`sample_annotation`
: data frame with:
  1. `sample_id_col` (this can be repeated as row names)
  2. biological covariates
  3. technical covariates (batches etc)

  . See `help("example_sample_annotation")`

`time_column`
: name of the column(s) where run date & time are specified. These will be used to determine the run order

`new_time_column`
: name of the new column that will contain the converted date/time value

`dateTimeFormat`
: POSIX format of the date and time. See `as.POSIXct` from base R for details

`new_order_col`
: name of the column containing the generated sample run order based on time columns

`instrument_col`
: name of the column denoting the instrument used for measurements

Value

sample annotation file with a new column `new_time_column` with POSIX-formatted date & `new_order_col` used in some diagnostic plots (e.g. `plot_iRT`, `plot_sample_mean`)

Examples

```
data("example_sample_annotation", package = "proBatch")
sample_annotation_wOrder <- date_to_sample_order(
    example_sample_annotation,
    time_column = c("RunDate", "RunTime"),
    new_time_column = "new_DateTime",
    dateTimeFormat = c("%b_%d", "%H:%M:%S"),
    new_order_col = "new_order",
    instrument_col = NULL
)
```

`define_sample_order` *Defining sample order internally*

Description

Defining sample order internally

Usage

```
define_sample_order(
  order_col,
  sample_annotation,
  facet_col,
  batch_col,
  df_long,
  sample_id_col,
  color_by_batch
)
```

Arguments

`order_col` column in `sample_annotation` that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be ‘NULL‘ if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")` facet_col — column in sample_annotation with a batch factor to separate plots into facets; usually 2nd to batch_col. Most meaningful for multi-instrument MS experiments (where each instrument has its own order-associated effects (see order_col) or simultaneous examination of two batch factors (e.g. preparation day and measurement day). For single-instrument case should be set to 'NULL'

batch_col — column in sample_annotation that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

df_long — data frame where each row is a single feature in a single sample. It minimally has a sample_id_col, a feature_id_col and a measure_col, but usually also an m_score (in OpenSWATH output result file). See help("example_proteome") for more details.

sample_id_col — name of the column in sample_annotation table, where the filenames (colnames of the data_matrix are found).

color_by_batch — (logical) whether to color points and connecting lines by batch factor as defined by batch_col.

Value

list of two items: order_col new name and new df_long

See Also

plot_sample_mean_or_boxplot, feature_level_diagnostics

Examples

```
data(list = c("example_proteome", "example_sample_annotation"), package = "proBatch")
sample_order <- define_sample_order(
    order_col = "order",
    sample_annotation = example_sample_annotation,
    facet_col = NULL, batch_col = "MS_batch", df_long = example_proteome,
    sample_id_col = "FullRunName", color_by_batch = TRUE
)
new_order_col <- sample_order$order_col
df_long <- sample_order$df_long
```

---

example_ecoli_data — *Example multi-center DIA LFQ E. coli proteomics (DIA-NN)*

Description

An example dataset illustrating a typical multi-center DIA (data-independent acquisition) proteomics study processed with DIA-NN. Subset of 300 proteins and 1119 precursors from the PRIDE ID PXD053812. Study design summary: - Five independent centers. - Per center: 10(9) samples of E.coli grown in different media (Pyruvate vs Glucose) The distributed object is a named list that contains the combined study-level tables derived from all centers (per-center sub-lists were removed to keep the package size below 5 MB).

Format

A named list with four elements:

‘all_metadata‘ ‘data.frame‘ (118 rows × 3 columns) with per-sample annotation. Columns: ‘Run‘, ‘Lab‘, and ‘Condition‘.

‘all_precursors‘ ‘data.frame‘ (1463 rows × 118 columns) containing the combined precursor-level Precursor.Normalised intensities (rows = precursors, columns = samples).

‘all_protein_groups‘ ‘data.frame‘ (400 rows × 118 columns) containing the combined protein group-level PG.MaxLFQ intensities.

‘all_precursor_pg_match‘ ‘data.frame‘ (1463 rows × 2 columns) linking ‘Precursor.Id‘ to ‘Protein.Ids‘.

Value

A named list containing the combined metadata, quantification matrices, and precursor-to-protein mapping.

Source

PRIDE ID PXD053812

Examples

```
data("example_ecoli_data", package = "proBatch")
names(example_ecoli_data)
```

`example_peptide_annotation`

Peptide annotation data

Description

This is data from Aging study annotated with gene names

Format

A data frame with 535 rows and 10 variables:

peptide_group_label peptide group label ID, identical to `peptide_group_label` in `example_proteome`

Gene HUGO gene ID

ProteinName protein group name as specified in `example_proteome`

Value

A data frame with 535 rows and 10 variables.

Examples

```
data("example_peptide_annotation", package = "proBatch")
head(example_peptide_annotation)
```

---

example_proteome *Example protein data in long format*

Description

This is OpenSWATH-output data from Aging study with all iRT, spike-in peptides, few representative peptides and proteins for signal improvement demonstration. Using `matrix_to_long` can be converted to `example_proteome_matrix`

Format

A data frame with 124655 rows and 7 variables:

peptide_group_label peptide ID, which is regular feature level. This column is mostly used as `feature_id_col`used for merging with `"example_peptide_annotation"`

Intensity peptide group intensity in given sample. Used in function as `measure_col`

Protein Protein group ID, specified as N/UniProtID1|UniProtID2|..., where N is number of protein peptide group maps to. If 1/UniProtID, then this is proteotypic peptide, in functions used as `protein_col`

FullRunName name of the file, in most functions used for `sample_id_col`

m_score column marking the quality of peptide IDs, used as `qual_col` throughout the script; when `qual_value` is 2 in this column, peptide has been imputed (requantified) ...

Value

A data frame with 124655 rows and 7 variables.

Source

PRIDE ID will be added upon the publication of the dataset

Examples

```
data("example_proteome", package = "proBatch")
head(example_proteome)
```

example_proteome_matrix
Example protein data in matrix

Description

This is measurement data from Aging study with columns representing samples and rows representing peptides. Generated by `long_to_matrix`

Format

A matrix with 535 rows and 233 columns:

Value

A matrix with 535 rows and 233 columns.

Source

PRIDE ID will be added upon the publication of the dataset

Examples

```
data("example_proteome_matrix", package = "proBatch")
dim(example_proteome_matrix)
```

example_sample_annotation
Sample annotation data version 1

Description

This is data from BXD mouse population aging study with mock instruments to show how instrument-specific functionality works

Format

A data frame with 233 rows and 11 variables:

FullRunName name of the file with the measurement for each sample, referred to as `sample_id_col`

MS_batch mass-spectrometry batch: 4-level factor of manually annotated batches

EarTag mouse ID, i.e. ID of the biological object. Only 14 mice have been replicated, one mouse was profiled 7 times.

Strain mouse strain ID from BXD population set - biological covariate #1, 51 Strain represented

Diet diet, biological covariate #2 - either `HFD` = ‘High Fat Diet‘ or `CD` = ‘Chow Diet‘

Sex mice sex - biological covariate #3

RunDate mass-spectrometry running date. In combination with `RunTime` used for running order determination. Vector of class `"difftime"` and `"hms"`

RunTime mass-spectrometry running time. In combination with `RunDate` used for running order determination.Vector of class `"POSIXct"` and `"POSIXt"`

DateTime numeric date and time generated by `date_to_sample_order`

order order of samples generated by sorting DateTime in `date_to_sample_order`

digestion_batch peptide digestion batch: 4-level factor of manually annotated batches ...

Value

A data frame with 233 rows and 11 variables.

Examples

```
data("example_sample_annotation", package = "proBatch")
head(example_sample_annotation)
```

`feature_level_diagnostics`

Plotting peptide measurements

Description

Creates a peptide faceted ggplot2 plot of the value in `measure_col` vs `order_col` (if ‘NULL‘, x-axis is simply a sample name order). Additionally, the resulting plot can also be colored either by batch factor, by quality factor (e.g. imputed/non-imputed) and, if needed, faceted by another batch factor, e.g. an instrument. If the non-linear curve was fit, this can also be added to the plot, see functions specific to each case below

Usage

```
plot_single_feature(
  feature_name,
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  feature_id_col = "peptide_group_label",
  geom = c("point", "line"),
  qual_col = NULL,
  qual_value = NULL,
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
```

```
  vline_color = "red",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme = "classic",
  ylimits = NULL,
  base_size = 20
)

plot_peptides_of_one_protein(
  protein_name,
  peptide_annotation = NULL,
  protein_col = "ProteinName",
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  feature_id_col = "peptide_group_label",
  geom = c("point", "line"),
  qual_col = NULL,
  qual_value = NULL,
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
  vline_color = "red",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = sprintf("Peptides of %s protein", protein_name),
  theme = "classic",
  base_size = 20
)

plot_spike_in(
  spike_ins = "BOVIN",
  peptide_annotation = NULL,
  protein_col = "ProteinName",
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  feature_id_col = "peptide_group_label",
```

```
  geom = c("point", "line"),
  qual_col = NULL,
  qual_value = NULL,
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
  vline_color = "red",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = sprintf("Spike-in %s plots", spike_ins),
  theme = "classic",
  base_size = 20
)

plot_iRT(
  irt_pattern = "iRT",
  peptide_annotation = NULL,
  protein_col = "ProteinName",
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  feature_id_col = "peptide_group_label",
  geom = c("point", "line"),
  qual_col = NULL,
  qual_value = NULL,
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
  vline_color = "red",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = "iRT peptide profile",
  theme = "classic",
  base_size = 20
)

plot_with_fitting_curve(
  feature_name,
  fit_df,
```

```
  fit_value_col = "fit",
  df_long,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  feature_id_col = "peptide_group_label",
  geom = c("point", "line"),
  qual_col = NULL,
  qual_value = NULL,
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
  vline_color = "grey",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
 plot_title = sprintf("Fitting curve of %s peptide", paste(feature_name, collapse =
    " ")),
  theme = "classic",
  base_size = 20
)
```

Arguments

`feature_name` name of the selected feature (e.g. peptide) for diagnostic profiling

`df_long` data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`measure_col` if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`geom` whether to show the feature as points and/or connect by lines (accepted values are: 1. `point`, `line` and `c('point', 'line')`)

`qual_col`	column to color point by certain value denoted by `qual_value`. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to `m_score`.
`qual_value`	value in `qual_col` to color. For OpenSWATH data, this argument value has to be set to `2` (this is an `m_score` value for imputed values (requant values).
`batch_col`	column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).
`color_by_batch`	(logical) whether to color points and connecting lines by batch factor as defined by `batch_col`.
`color_scheme`	a named vector of colors to map to `batch_col`, names corresponding to the levels of the factor. For continuous variables, vector doesn't need to be named.
`order_col`	column in `sample_annotation` that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order
`vline_color`	color of vertical lines, typically separating different MS batches in ordered runs; should be 'NULL' for experiments without intrinsic order
`facet_col`	column in `sample_annotation` with a batch factor to separate plots into facets; usually 2nd to `batch_col`. Most meaningful for multi-instrument MS experiments (where each instrument has its own order-associated effects (see `order_col`) or simultaneous examination of two batch factors (e.g. preparation day and measurement day). For single-instrument case should be set to 'NULL'
`filename`	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
`width`	option determining the output image width
`height`	option determining the output image height
`units`	units: 'cm', 'in' or 'mm'
`plot_title`	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
`theme`	ggplot theme, by default `classic`. Can be easily overriden
`ylimits`	range of y-axis to plot feature-level trends
`base_size`	base font size
`protein_name`	name of the protein as defined in `ProteinName`
`peptide_annotation`	long format data frame with peptide ID and their corresponding protein and/or gene annotations. See `help("example_peptide_annotation")`.
`protein_col`	column where protein names are specified
`spike_ins`	name of feature(s), typically proteins that were spiked in for control
`irt_pattern`	substring used to identify iRT proteins in the column 'ProteinName'
`fit_df`	data frame output of `adjust_batch_trend_df` to be plotted with the line
`fit_value_col`	column in `fit_df` where the values for fitting trend are found

Value

ggplot2 type plot of `measure_col` vs `order_col`, faceted by `feature_name` and (optionally) by `batch_col`

Examples

```
data(list = c(
    "example_sample_annotation", "example_proteome",
    "example_peptide_annotation"
), package = "proBatch")

sample_annotation <- example_sample_annotation
peptide_annotation <- example_peptide_annotation
proteome <- example_proteome

feature_id <- "10231_QDVDVWLWQQEGSSK_2"

feature_plot <- plot_single_feature(
    feature_name = feature_id,
    df_long = proteome,
    sample_annotation = sample_annotation,
    color_by_batch = TRUE,
    batch_col = "MS_batch"
)

protein_plot <- plot_peptides_of_one_protein(
    protein_name = "Haao",
    peptide_annotation = peptide_annotation,
    df_long = proteome,
    sample_annotation = sample_annotation,
    protein_col = "Gene"
)

spike_in_plot <- plot_spike_in(
    spike_ins = "BOVINE_A1ag",
    peptide_annotation = peptide_annotation,
    df_long = proteome,
    sample_annotation = sample_annotation,
    protein_col = "Gene"
)

irt_plot <- plot_iRT(
    irt_pattern = "iRT",
    peptide_annotation = peptide_annotation,
    df_long = proteome,
    sample_annotation = sample_annotation,
    protein_col = "Gene"
)

fit_input <- adjust_batch_trend_df(
    proteome[proteome$peptide_group_label == feature_id, ],
    sample_annotation,
```

```
    span = 0.7
)

curve_plot <- plot_with_fitting_curve(
    feature_name = feature_id,
    df_long = proteome,
    sample_annotation = sample_annotation,
    fit_df = fit_input
)
```

`fit_nonlinear` *Fit a non-linear trend (currently optimized for LOESS)*

Description

Fit a non-linear trend (currently optimized for LOESS)

Usage

```
fit_nonlinear(
  df_feature_batch,
  measure_col = "Intensity",
  order_col = "order",
  feature_id = NULL,
  batch_id = NULL,
  fit_func = "loess_regression",
  optimize_span = FALSE,
  no_fit_imputed = TRUE,
  qual_col = "m_score",
  qual_value = 2,
  min_measurements = 8,
  ...
)
```

Arguments

`df_feature_batch`
: data frame containing response variable e.g. samples in order and explanatory variable e.g. measurement for a specific feature (peptide) in a specific batch

`measure_col`
: if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

`order_col`
: column in `sample_annotation` that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

`feature_id`	the name of the feature, required for warnings
`batch_id`	the name of the batch, required for warnings
`fit_func`	function to use for the fit, e.g. `loess_regression`
`optimize_span`	logical, whether to specify span or optimize it (specific entirely for LOESS regression)
`no_fit_imputed`	(logical) whether to fit the imputed (requant) values
`qual_col`	column to color point by certain value denoted by `qual_value`. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to `m_score`.
`qual_value`	value in `qual_col` to color. For OpenSWATH data, this argument value has to be set to `2` (this is an `m_score` value for imputed values (requant values).
`min_measurements`	the absolute threshold to filter
`...`	additional parameters to be passed to the fitting function

Value

vector of fitted response values

Examples

```
# Load necessary datasets
data(list = c("example_proteome", "example_sample_annotation"), package = "proBatch")

test_peptide <- example_proteome$peptide_group_label[1]
selected_peptide <- example_proteome$peptide_group_label == test_peptide
df_selected <- example_proteome[selected_peptide, ]
selected_batch <- example_sample_annotation$MS_batch == "Batch_1"
batch_selected_df <- example_sample_annotation[selected_batch, ]
df_for_test <- merge(df_selected, batch_selected_df, by = "FullRunName")
fit_values <- fit_nonlinear(df_for_test)

# for the case where are two many missing values, no curve is fit
selected_batch <- example_sample_annotation$MS_batch == "Batch_2"
batch_selected_df <- example_sample_annotation[selected_batch, ]
df_for_test <- merge(df_selected, batch_selected_df, by = "FullRunName")
fit_values <- fit_nonlinear(df_for_test)
missing_values <- df_for_test[["m_score"]] == 2
all(fit_values[!is.na(fit_values)] == df_for_test[["Intensity"]][!missing_values])
```

`generate_colors_for_numeric` *Generates color vector from continous palette*

Description

Generates a vector of colors for a vector of numeric, POSIXct (i.e. the (signed) number of seconds since the beginning of 1970 , or factors

Usage

```
generate_colors_for_numeric(palette_type = "brewer", i = 1)
```

Arguments

`palette_type`	'brewer' or 'viridis'
`i`	if `palette_type` is 'brewer' the palette argument to `brewer_pal`. If `palette_type` is 'viridis' the option argument to `viridis_pal`

Value

vector of colors

Examples

```
generate_colors_for_numeric("brewer", i = 1)
```

`get_chain` *Retrieve operation chain as vector or single string "combat_on_mediannorm_on_log"*

Description

Retrieve operation chain as vector or single string "combat_on_mediannorm_on_log"

Usage

```
get_chain(object, as_string = FALSE)
```

Arguments

`object`	A 'ProBatchFeatures' object.
`as_string`	logical(1). if 'TRUE' returns the chain as a single string of the form '"combat_on_mediannorm_on_log"'.

Value

Character vector or string describing the processing chain.

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object ---------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ----------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
```

```
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`get_operation_log` *Access the operation log (structured)*

Description

Access the operation log (structured)

Usage

```
get_operation_log(object)
```

Arguments

`object`	A ‘ProBatchFeatures‘ object.

Value

S4Vectors::DataFrame

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object ----------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
```

```
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ---------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`guess_factor_columns_if_needed`

Guess factors if numeric columns were not provided

Description

Derive numeric columns from factor columns if guess_factors is TRUE and numeric columns are NULL.

Usage

```
guess_factor_columns_if_needed(
  factor_columns,
  sample_annotation,
  guess_factors
)
```

Arguments

`factor_columns` character vector of factor columns.

`sample_annotation` data frame of sample annotations.

`guess_factors` logical indicating whether to guess numeric columns.

Value

Named list containing updated factor_columns and numeric_columns.

---

handle_factor_numeric_overlap

Handle factor columns that are duplicated in numeric_columns

Description

Remove numeric columns from factor columns if overlap is detected.

Usage

```
handle_factor_numeric_overlap(factor_columns, numeric_columns)
```

Arguments

`factor_columns` character vector of factor columns.

`numeric_columns` character vector of numeric columns.

Value

List with updated factor_columns and a warning if overlaps exist

handle_missing_values *Handle missing values in a data matrix*

Description

This function can either fill missing values with a specified value or remove rows (and columns, if applicable) with missing values. It is primarily intended for use prior to batch correction methods that cannot handle missing values, such as ComBat or limma's removeBatchEffect, or plotting functions that require complete data.

Usage

```
handle_missing_values(data_matrix, warning_message, fill_the_missing = NULL)
```

Arguments

`data_matrix` A numeric matrix with features in rows and samples in columns.

`warning_message` A character string with a warning shown if missing values are found.

`fill_the_missing` A control value: - FALSE: do nothing (keep NAs). - Missing (arg not supplied) or "remove"/"rm"/"REMOVE": remove rows with any NA (and matching columns if square & symmetric). - Numeric scalar: fill NAs with this value. - Non-numeric: coerced to 0 with a warning and used to fill NAs.

Details

Semantics: - If there are no NAs: return input unchanged. - If fill_the_missing is explicitly FALSE: do nothing (keep NAs). - If fill_the_missing is missing (argument not supplied) or one of "remove","rm","REMOVE": remove rows with any NA; if the matrix is square and symmetric (na.rm=TRUE), remove matching rows AND columns using the same row keep-mask. - Otherwise: if non-numeric or NA, coerce to 0 with a warning; then fill NAs.

Value

A matrix with missing values handled as specified.

Examples

```
mat <- matrix(c(1, NA, 3, 4), nrow = 2)
suppressWarnings(proBatch:::handle_missing_values(
    mat,
    warning_message = "demo",
    fill_the_missing = 0
))
```

long_to_matrix *Long to wide data format conversion*

Description

Convert from a long data frame representation to a wide matrix representation

Usage

```
long_to_matrix(
  df_long,
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  sample_id_col = "FullRunName",
  qual_col = NULL,
  qual_value = 2
)
```

Arguments

`df_long`	data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.
`feature_id_col`	name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.
`measure_col`	if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.
`sample_id_col`	name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).
`qual_col`	column to color point by certain value denoted by `qual_value`. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to `m_score`.
`qual_value`	value in `qual_col` to color. For OpenSWATH data, this argument value has to be set to `2` (this is an `m_score` value for imputed values (requant values).

Value

`data_matrix` (proBatch) like matrix (features in rows, samples in columns)

See Also

Other matrix manipulation functions: `matrix_to_long()`

Examples

```
data("example_proteome", package = "proBatch")
proteome_matrix <- long_to_matrix(example_proteome)
```

`matrix_to_long` *Wide to long conversion*

Description

Convert from wide matrix to a long data frame representation

Usage

```
matrix_to_long(
  data_matrix,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  measure_col = "Intensity",
  sample_id_col = "FullRunName",
  step = NULL
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`measure_col` if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`step` normalization step (e.g. `Raw` or `Normalized`. Useful if consecutive steps are compared in plots. Note that in plots these are usually ordered alphabetically, so it's worth naming with numbers, e.g. `1_raw`, `2_quantile`

Value

df_long (proBatch) like data frame

See Also

Other matrix manipulation functions: long_to_matrix()

Examples

```
# Load necessary datasets
data(
    list = c("example_sample_annotation", "example_proteome_matrix"),
    package = "proBatch"
)
# Convert matrix to long format
proteome_long <- matrix_to_long(
    example_proteome_matrix,
    example_sample_annotation
)
```

---

merge_rare_levels *Replaces rare levels with other*

Description

Replaces levels with a maximal occurrence of 1 with other

Usage

```
merge_rare_levels(column, rare_thr = 2)
```

Arguments

column	column of the data whose rare categories need to be merged to "other"
rare_thr	minimal number of times for a category to be represented to be declared as "rare" and converted to "other"

Value

column with rare occurrences replaced by other

Examples

```
column <- factor(c("A", "B", "A", "C", "D", "D", "E"))
merge_rare_levels(column, rare_thr = 2)
# [1] A     other A     other D     D     other
# Levels: A D other
```

normalize *Data normalization methods*

Description

Normalization of raw (usually log-transformed) data. Normalization brings the samples to the same scale. Currently the following normalization functions are implemented: #'

1. Quantile normalization: 'quantile_normalize_dm()'. Quantile normalization of the data.
2. Median normalization: 'normalize_sample_medians_dm()'. Normalization by centering sample medians to global median of the data

Alternatively, one can call normalization function with 'normalize_data_dm()' wrapper.

Usage

```
quantile_normalize_dm(data_matrix)

quantile_normalize_df(
  df_long,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = 2,
  keep_all = "default"
)

normalize_sample_medians_dm(data_matrix)

normalize_sample_medians_df(
  df_long,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  no_fit_imputed = FALSE,
  qual_col = NULL,
  qual_value = 2,
  keep_all = "default"
)

normalize_data_dm(
  data_matrix,
  normalize_func = c("quantile", "medianCentering"),
  log_base = NULL,
  offset = 1
```

```
)

normalize_data_df(
  df_long,
  normalize_func = c("quantile", "medianCentering"),
  log_base = NULL,
  offset = 1,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  no_fit_imputed = TRUE,
  qual_col = NULL,
  qual_value = 2,
  keep_all = "default"
)
```

Arguments

- `data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)
- `df_long` data frame where each row is a single feature in a single sample. It minimally has a `sample_id_col`, a `feature_id_col` and a `measure_col`, but usually also an `m_score` (in OpenSWATH output result file). See `help("example_proteome")` for more details.
- `feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.
- `sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).
- `measure_col` if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.
- `no_fit_imputed` (logical) whether to use imputed (requant) values, as flagged in `qual_col` by `qual_value` for data transformation
- `qual_col` column to color point by certain value denoted by `qual_value`. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to `m_score`.
- `qual_value` value in `qual_col` to color. For OpenSWATH data, this argument value has to be set to `2` (this is an `m_score` value for imputed values (requant values).
- `keep_all` when transforming the data (normalize, correct) - acceptable values: all/default/minimal (which set of columns be kept).
- `normalize_func` global batch normalization method (‘quantile‘ or ‘MedianCentering‘)
- `log_base` whether to log transform data matrix before normalization (e.g. ‘NULL‘, ‘2‘ or ‘10‘)
- `offset` small positive number to prevent 0 conversion to `-Inf`

Value

the data in the same format as input (`data_matrix` or `df_long`). For `df_long` the data frame stores the original values of `measure_col` in another column called "preNorm_intensity" if "intensity", and the normalized values in `measure_col` column.

Examples

```
data(list = c("example_proteome", "example_proteome_matrix"), package = "proBatch")

# Quantile normalization:
quantile_normalized_matrix <- quantile_normalize_dm(example_proteome_matrix)

# Median centering:
median_normalized_df <- normalize_sample_medians_df(example_proteome)

# Transform the data in one go:
quantile_normalized_matrix <- normalize_data_dm(example_proteome_matrix,
    normalize_func = "quantile", log_base = 2, offset = 1
)
```

---

pb_add_level — *Add a new level from an external matrix and link to an existing assay*

Description

Add a new level from an external matrix and link to an existing assay

Usage

```
pb_add_level(
  object,
  from,
  new_matrix,
  to_level,
  to_pipeline = NULL,
  name = NULL,
  mapping_df = NULL,
  from_id = NULL,
  to_id = NULL,
  map_strategy = c("as_is", "first", "longest"),
  link_var = "ProteinID",
  backend = c("auto", "memory", "hdf5"),
  hdf5_path = NULL
)
```

Arguments

`object`	ProBatchFeatures
`from`	assay name (e.g., "peptide::raw")
`new_matrix`	numeric matrix (features x samples)
`to_level`	e.g. "protein"
`to_pipeline`	optional pipeline name (default carries over from 'from')
`name`	optional final assay name override
`mapping_df`	data.frame with mapping from 'from' IDs to 'to' IDs
`from_id`	column in mapping_df for 'from' IDs (e.g., "Precursor.Id")
`to_id`	column in mapping_df for 'to' IDs (e.g., "Protein.Ids")
`map_strategy`	how to resolve multiple to-ids per from-id: "as_is" (error if not 1:1), "first" (take first), "longest" (take longest string)
`link_var`	rowData variable name to use for linking (e.g., "ProteinID")
`backend`	"memory","hdf5","auto"
`hdf5_path`	optional filepath for HDF5Array

Value

ProBatchFeatures with new assay and link added

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object -------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it ----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
```

```
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ---------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ---------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)
```

```
# Aggregate and add levels -------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_aggregate_level` *Aggregate features (e.g., peptide -> protein) and store as new level*

Description

Aggregate features (e.g., peptide -> protein) and store as new level

Usage

```
pb_aggregate_level(
  object,
  from,
  feature_var,
  fun = colMedians,
  new_level = "protein",
  new_pipeline = NULL
)
```

Arguments

`object`	ProBatchFeatures
`from`	assay name (e.g., "peptide::raw")
`feature_var`	name of a column in rowData(from) holding group labels (e.g. protein IDs)
`fun`	summarization function (e.g., matrixStats::colMedians), or name
`new_level`	new level label (e.g., "protein")
`new_pipeline`	optional pipeline name (default carries over from 'from')

Value

ProBatchFeatures with an additional aggregated assay appended

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
```

```
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object -----------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor --------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
```

```
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_assay_matrix` *Convenience accessor for assay matrix by name/index (returns the 'intensity' assay)*

Description

Convenience accessor for assay matrix by name/index (returns the 'intensity' assay)

Usage

```
pb_assay_matrix(object, assay = NULL, name = "intensity")
```

Arguments

object	A ‘ProBatchFeatures‘ object.
assay	Assay identifier to extract; defaults to the current assay.
name	Assay entry to read from the underlying ‘SummarizedExperiment‘.

Value

assay data matrix with features in rows and samples in columns

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
```

```
))

# Long-format constructor ---------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_as_long` *Get current assay as LONG (via proBatch::matrix_to_long)*

Description

Get current assay as LONG (via proBatch::matrix_to_long)

Usage

```
pb_as_long(
  object,
  feature_id_col = "feature_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  pbf_name = pb_current_assay(object)
)
```

Arguments

`object`	A 'ProBatchFeatures' object.
`feature_id_col`	Column name used for feature identifiers in the long table.
`sample_id_col`	Column name used for sample identifiers in the long table.
`measure_col`	Column name containing measured values in the long table.
`pbf_name`	Assay name whose intensities should be returned in long form.

Value

tibble/data.frame containing one row per feature-sample combination

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
```

```
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels ---------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

pb_as_wide *Get an assay matrix (wide)*

Description

Get an assay matrix (wide)

Usage

```
pb_as_wide(object, assay = pb_current_assay(object), name = "intensity")
```

Arguments

object	A ‘ProBatchFeatures‘ object.
assay	Assay identifier to extract; defaults to the current assay.
name	Assay entry name inside the ‘SummarizedExperiment‘ to return.

Value

numeric matrix (wide) corresponding to the requested assay

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)
```

```
# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices --------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels ---------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_current_assay` *Current (latest) assay name*

Description

Current (latest) assay name

Usage

```
pb_current_assay(object)
```

Arguments

object	A ‘ProBatchFeatures‘ object.

Value

character(1) assay identifier for the most recently stored assay

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object -------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it ----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers ----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
```

```
    store_fast_steps = TRUE
)

# Get information about the object ----------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices ----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor -------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels ------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
```

```
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_eval` *Evaluate a pipeline and return the matrix, without storing*

Description

Evaluate a pipeline and return the matrix, without storing

Usage

```
pb_eval(object, from, steps, funs = NULL, params_list = NULL)
```

Arguments

`object`	ProBatchFeatures
`from`	assay name (e.g., "peptide::raw")
`steps`	character vector, e.g. c("log2","medianNorm","combat")
`funs`	optional same-length vector/list of functions/names (default: steps for registry lookup)
`params_list`	list of parameter lists (same length as steps)

Value

numeric matrix (features x samples)

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
```

```
)

# Register a custom step and evaluate it ----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers ----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object --------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing -------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ---------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
```

```
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

---

pb_missing_helpers   *Apply 'QFeatures' missing-data helpers to stored assays*

Description

These wrappers delegate to the corresponding ‘QFeatures‘ generics while ensuring that the requested assays remain part of the ‘ProBatchFeatures‘ object. Only assays that are already materialised can be modified. If a transformation step was applied as a "fast" step (log, log2, etc.), consider re-running it with ‘store_fast_steps = TRUE‘.

Usage

```
pb_zeroIsNA(object, pbf_name = names(object), ...)

pb_infIsNA(object, pbf_name = names(object), ...)

pb_nNA(object, pbf_name = names(object), ...)

pb_filterNA(object, pbf_name = NULL, inplace = FALSE, final_name = NULL, ...)
```

Arguments

object	A ‘ProBatchFeatures‘ object.
pbf_name	Character vector of assay names. Defaults to ‘names(object)‘ - all assays.
...	Additional parameters forwarded to the underlying ‘QFeatures‘ method where applicable.
inplace	Logical (used by ‘pb_filterNA()‘ only), whether to modify the object in place. Default: ‘FALSE‘. If ‘FALSE‘, the modified assay(s) will be added to the object with ‘final_name‘ (if provided) or the original name(s) with suffix ‘_filteredNA‘.
final_name	Character (used by ‘pb_filterNA()‘ only), name for the modified assay(s) if ‘inplace‘ is ‘FALSE‘. If ‘NULL‘ (default), the original name(s) with suffix ‘_filteredNA‘ will be used.

Value

'pb_zeroIsNA()', 'pb_infIsNA()' and 'pb_filterNA()' return the updated 'ProBatchFeatures' object. 'pb_nNA()' returns the output of the corresponding 'QFeatures::nNA()' call (a 'list' of 'DataFrame's).

---

pb_pipeline_name *Pretty pipeline name derived from the assay*

Description

Pretty pipeline name derived from the assay

Usage

```
pb_pipeline_name(object, assay = pb_current_assay(object))
```

Arguments

object	ProBatchFeatures
assay	character(1) assay name; defaults to current assay

Value

character(1) pipeline string like "combat_on_medianNorm_on_log2" or "raw"

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object -------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))
```

```
# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
```

```
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_register_step` *Allow to register/override steps at runtime (e.g., map "combat" -> proBatch::combat_dm)*

Description

Allow to register/override steps at runtime (e.g., map "combat" -> proBatch::combat_dm)

Usage

```
pb_register_step(name, fun)
```

Arguments

name	character(1) step name
fun	function implementing the step

Value

NULL (invisible)

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
```

```
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
```

```
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels -------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`pb_transform` *Compute a pipeline and optionally store only the final result*

Description

Compute a pipeline and optionally store only the final result

Usage

```
pb_transform(
  object,
  from,
  steps,
  funs = NULL,
  params_list = NULL,
  level = NULL,
  store_fast_steps = FALSE,
  fast_steps = c("log", "log2", "medianNorm"),
  store_intermediate = FALSE,
  final_name = NULL,
  backend = c("auto", "memory", "hdf5"),
  hdf5_path = NULL
)
```

Arguments

`object`	A ‘ProBatchFeatures‘ object.
`from`	Assay name to start the pipeline from.
`steps`	character vector, e.g. c("log2","medianNorm","combat")
`funs`	optional same-length vector/list of functions/names (default: steps)

`params_list`	list of parameter lists (same length as steps)
`level`	Optional level label to assign to the generated assay(s).
`store_fast_steps`	logical; if FALSE, fast steps are computed but not stored
`fast_steps`	which steps count as fast (default: c("log","log2","medianNorm"))
`store_intermediate`	logical; if TRUE store every step (overrides fast behavior)
`final_name`	optional final assay name override
`backend`	"memory","hdf5","auto"
`hdf5_path`	Optional file path used when ‘backend = "hdf5"‘.

Value

ProBatchFeatures with the requested pipeline added (as log and/or assay)

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
```

```
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor --------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels -------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

plot_corr_matrix *Visualise correlation matrix*

Description

recommended for heatmap-type visualisation of correlation matrix with <100 items. With >50 samples and ~10 replicate pairs distribution plots may be more informative.

Usage

```
plot_corr_matrix(
  corr_matrix,
  annotation = NULL,
  annotation_id_col = "FullRunName",
  factors_to_plot = NULL,
  cluster_rows = FALSE,
  cluster_cols = FALSE,
  heatmap_color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100),
  color_list = NULL,
  filename = NULL,
  width = 7,
  height = 7,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  ...
)
```

Arguments

`corr_matrix`	square correlation matrix
`annotation`	data frame with `peptide_annotation` for protein correlation heatmap or `sample_annotation` for sample correlation heatmap
`annotation_id_col`	`feature_id_col` for protein correlation heatmap or `sample_id_col` for sample correlation heatmap
`factors_to_plot`	vector of technical and biological covariates to be plotted in this diagnostic plot (assumed to be present in `sample_annotation`)
`cluster_rows`	boolean values determining if rows should be clustered or `hclust` object
`cluster_cols`	boolean values determining if columns should be clustered or `hclust` object
`heatmap_color`	vector of colors used in heatmap.
`color_list`	list, as returned by `sample_annotation_to_colors`, where each item contains a color vector for each factor to be mapped to the color.

filename	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
width	option determining the output image width
height	option determining the output image height
units	units: 'cm', 'in' or 'mm'
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
...	parameters for the pheatmap visualisation, for details see examples and help to corresponding functions

Details

Plot correlation of selected samples or peptides

Value

pheatmap object

See Also

pheatmap, plot_sample_corr_distribution, plot_peptide_corr_distribution

Examples

```
data("example_proteome_matrix", package = "proBatch")
peptides <- c("10231_QDVDVWLWQQEGSSK_2", "10768_RLESELDGLR_2")
data_matrix_sub <- example_proteome_matrix[peptides, ]
corr_matrix <- cor(t(data_matrix_sub), use = "complete.obs")
corr_matrix_plot <- plot_corr_matrix(corr_matrix)
```

---

plot_CV_distr *Plot CV distribution to compare various steps of the analysis*

Description

Plot CV distribution to compare various steps of the analysis

Usage

```
plot_CV_distr(
  df_long,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
```

```
    biospecimen_id_col = "EarTag",
    batch_col = NULL,
    unlog = TRUE,
    log_base = 2,
    offset = 1,
    plot_title = NULL,
    filename = NULL,
    theme = "classic"
  )
```

Arguments

df_long	as in df_long for the rest of the package, but, when it has entries for intensity, represented in measure_col for several steps, e.g. raw, normalized, batch corrected data, as seen in column Step, then multi-step CV comparison can be carried out.
sample_annotation	data frame with: 1. sample_id_col (this can be repeated as row names) 2. biological covariates 3. technical covariates (batches etc) . See help("example_sample_annotation")
feature_id_col	name of the column with feature/gene/peptide/protein ID used in the long format representation df_long. In the wide formatted representation data_matrix this corresponds to the row names.
sample_id_col	name of the column in sample_annotation table, where the filenames (colnames of the data_matrix are found).
measure_col	if df_long is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.
biospecimen_id_col	column in sample_annotation that defines a unique bio ID, which is usually a combination of conditions or groups. Tip: if such ID is absent, but can be defined from several columns, create new biospecimen_id column
batch_col	column in sample_annotation that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).
unlog	(logical) whether to reverse log transformation of the original data
log_base	base of the logarithm for transformation
offset	small positive number to prevent 0 conversion to -Inf
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
filename	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
theme	ggplot theme, by default classic. Can be easily overriden

Value

ggplot object with the boxplot of CVs on one or several steps

Examples

```
data(list = c("example_sample_annotation", "example_proteome"), package = "proBatch")
CV_plot <- plot_CV_distr(example_proteome,
    sample_annotation = example_sample_annotation,
    measure_col = "Intensity", batch_col = "MS_batch",
    plot_title = NULL, filename = NULL, theme = "classic"
)
```

---

plot_CV_distr.df — *Plot the distribution (boxplots) of per-batch per-step CV of features*

Description

Plot the distribution (boxplots) of per-batch per-step CV of features

Usage

```
plot_CV_distr.df(
  CV_df,
  plot_title = NULL,
  filename = NULL,
  theme = "classic",
  log_y_scale = TRUE
)
```

Arguments

`CV_df`	data frame with Total CV for each feature & (optionally) per-batch CV
`plot_title`	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
`filename`	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
`theme`	ggplot theme, by default `classic`. Can be easily overriden
`log_y_scale`	(logical) whether to display the CV on log-scale

Value

ggplot object

Examples

```
cv_example <- data.frame(
    Step = c("raw", "raw", "raw"),
    CV_total = c(10, 15, 12)
)
plot_CV_distr.df(cv_example, log_y_scale = FALSE)
```

---

plot_heatmap_diagnostic *Plot the heatmap of samples (cols) vs features (rows)*

Description

Plot the heatmap of samples (cols) vs features (rows)

Usage

```
## Default S3 method:
plot_heatmap_diagnostic(
  data_matrix,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  factors_to_plot = NULL,
  fill_the_missing = -1,
  color_for_missing = "black",
  heatmap_color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100),
  cluster_rows = TRUE,
  cluster_cols = FALSE,
  color_list = NULL,
  peptide_annotation = NULL,
  feature_id_col = NULL,
  factors_of_feature_ann = NULL,
  color_list_features = NULL,
  filename = NULL,
  width = 7,
  height = 7,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_heatmap_diagnostic(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
```

```
  peptide_annotation = NULL,
  feature_id_col = "peptide_group_label",
  plot_title = NULL,
  return_gridExtra = FALSE,
  plot_ncol = NULL,
  ...
)
```

Arguments

`data_matrix` Input object: matrix-like data or a 'ProBatchFeatures' instance.

`sample_annotation`
: data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`factors_to_plot`
: vector of technical and biological factors to be plotted in this diagnostic plot (assumed to be present in `sample_annotation`)

`fill_the_missing`
: numeric value that the missing values are substituted with, or `NULL` if features with missing values are to be excluded.

`color_for_missing`
: special color to make missing values. Usually black or white, depending on `heatmap_color`

`heatmap_color` vector of colors used in heatmap (typicall a gradient)

`cluster_rows` boolean value determining if rows should be clustered

`cluster_cols` boolean value determining if columns should be clustered

`color_list` list, as returned by `sample_annotation_to_colors`, where each item contains a color vector for each factor to be mapped to the color.

`peptide_annotation`
: long format data frame with peptide ID and their corresponding protein and/or gene annotations. See `help("example_peptide_annotation")`.

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`factors_of_feature_ann`
: vector of factors that characterize features, as listed in `peptide_annotation`

`color_list_features`
: list, as returned by `sample_annotation_to_colors`, but mapping `peptide_annotation` where each item contains a color vector for each factor to be mapped to the color.

filename	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
width	option determining the output image width
height	option determining the output image height
units	units: 'cm', 'in' or 'mm'
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
...	other parameters of `link[pheatmap]{pheatmap}`
pbf_name	Assay name(s) used when 'data_matrix' is a 'ProBatchFeatures'.
return_gridExtra	Logical; return arranged grobs instead of a plot list.
plot_ncol	Number of columns when arranging multiple assay plots.

Value

object returned by `link[pheatmap]{pheatmap}`

See Also

sample_annotation_to_colors, pheatmap

Examples

```
# Load necessary datasets
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")

# Use a smaller subset for the example
example_proteome_matrix_small <- example_proteome_matrix[1:50, ]

log_transformed_matrix <- log_transform_dm(example_proteome_matrix_small)
color_list <- sample_annotation_to_colors(example_sample_annotation,
    factor_columns = c(
        "MS_batch", "EarTag", "Strain",
        "Diet", "digestion_batch", "Sex"
    ),
    numeric_columns = c("DateTime", "order")
)

log_transformed_matrix <- log_transform_dm(example_proteome_matrix)
heatmap_plot <- plot_heatmap_diagnostic(log_transformed_matrix,
    example_sample_annotation,
    factors_to_plot = c("MS_batch", "digestion_batch", "Diet", "DateTime"),
    cluster_cols = TRUE, cluster_rows = FALSE,
    color_list = color_list, # can be NULL
    show_rownames = FALSE, show_colnames = FALSE
)
```

plot_heatmap_generic *Plot the heatmap*

Description

Plot the heatmap

Usage

```
## Default S3 method:
plot_heatmap_generic(
  data_matrix,
  column_annotation_df = NULL,
  row_annotation_df = NULL,
  col_ann_id_col = NULL,
  row_ann_id_col = NULL,
  columns_for_cols = c("MS_batch", "Diet", "DateTime", "order"),
  columns_for_rows = c("KEGG_pathway", "WGCNA_module", "evolutionary_distance"),
  cluster_rows = FALSE,
  cluster_cols = TRUE,
  annotation_color_cols = NULL,
  annotation_color_rows = NULL,
  fill_the_missing = -1,
  color_for_missing = "black",
  heatmap_color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100),
  filename = NULL,
  width = 7,
  height = 7,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_heatmap_generic(
  data_matrix,
  pbf_name = NULL,
  column_annotation_df = NULL,
  row_annotation_df = NULL,
  col_ann_id_col = NULL,
  row_ann_id_col = NULL,
  plot_title = NULL,
  return_gridExtra = FALSE,
  plot_ncol = NULL,
  ...
)
```

Arguments

data_matrix Input object: matrix-like data or a ‘ProBatchFeatures‘ instance.

column_annotation_df
data frame annotating columns of data_matrix

row_annotation_df
data frame annotating rows of data_matrix

col_ann_id_col column of column_annotation_df whose values are unique identifiers of columns in data_matrix

row_ann_id_col column of row_annotation_df whose values are unique identifiers of rows in data_matrix

columns_for_cols
vector of factors (columns) of column_annotation_df that will be mapped to color annotation of heatmap columns

columns_for_rows
vector of factors (columns) of row_annotation_df that will be mapped to color annotation of heatmap rows

cluster_rows boolean: whether the rows should be clustered

cluster_cols boolean: whether the rows should be clustered

annotation_color_cols
list of color vectors for column annotation, for each factor to be plotted; for factor-like variables a named vector (names should correspond to the levels of factors). Advisable to supply here color list returned by sample_annotation_to_colors

annotation_color_rows
list of color vectors for row annotation, for each factor to be plotted; for factor-like variables a named vector (names should correspond to the levels of factors). Advisable to supply here color list returned by sample_annotation_to_colors

fill_the_missing
numeric value that the missing values are substituted with, or NULL if features with missing values are to be excluded.

color_for_missing
special color to make missing values. Usually black or white, depending on heatmap_color

heatmap_color vector of colors used in heatmap (typicall a gradient)

filename path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

width option determining the output image width

height option determining the output image height

units units: ’cm’, ’in’ or ’mm’

plot_title title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

... other parameters of link[pheatmap]{pheatmap}

pbf_name Assay name(s) used when ‘data_matrix‘ is a ‘ProBatchFeatures‘.

return_gridExtra
: Logical; return arranged grobs instead of a plot list.

plot_ncol
: Number of columns when arranging multiple assay plots.

Value

pheatmap-type object

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
p <- plot_heatmap_generic(log_transform_dm(example_proteome_matrix),
    column_annotation_df = example_sample_annotation,
    columns_for_cols = c("MS_batch", "digestion_batch", "Diet", "DateTime"),
    plot_title = "test_heatmap",
    show_rownames = FALSE, show_colnames = FALSE
)
```

---

plot_hierarchical_clustering

cluster the data matrix to visually inspect which confounder dominates

---

Description

cluster the data matrix to visually inspect which confounder dominates

Usage

```
## Default S3 method:
plot_hierarchical_clustering(
  data_matrix,
  sample_annotation,
  sample_id_col = "FullRunName",
  color_list = NULL,
  factors_to_plot = NULL,
  fill_the_missing = 0,
  distance = "euclidean",
  agglomeration = "complete",
  label_samples = TRUE,
  label_font = 0.2,
  filename = NULL,
  width = 38,
  height = 25,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  ...
)
```

```
## S3 method for class 'ProBatchFeatures'
plot_hierarchical_clustering(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  plot_title = NULL,
  ...
)
```

Arguments

data_matrix — Input object: matrix-like data or a ‘ProBatchFeatures‘ instance.

sample_annotation — data frame with:
1. sample_id_col (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See help("example_sample_annotation")

sample_id_col — name of the column in sample_annotation table, where the filenames (colnames of the data_matrix are found).

color_list — list, as returned by sample_annotation_to_colors, where each item contains a color vector for each factor to be mapped to the color.

factors_to_plot — vector of technical and biological covariates to be plotted in this diagnostic plot (assumed to be present in sample_annotation)

fill_the_missing — numeric value determining how missing values should be substituted. If NULL, features with missing values are excluded.

distance — distance metric used for clustering

agglomeration — agglomeration methods as used by hclust

label_samples — if TRUE sample IDs (column names of data_matrix) will be printed

label_font — size of the font. Is active if label_samples is TRUE, ignored otherwise

filename — path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

width — option determining the output image width

height — option determining the output image height

units — units: ’cm’, ’in’ or ’mm’

plot_title — title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

... — other parameters of plotDendroAndColors from WGCNA package

pbf_name — Assay name(s) used when ‘x‘ is a ‘ProBatchFeatures‘.

Value

No return

See Also

hclust, sample_annotation_to_colors, plotDendroAndColors

Examples

```
# Load necessary datasets
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")

selected_batches <- example_sample_annotation$MS_batch %in%
    c("Batch_1", "Batch_2")
selected_samples <- example_sample_annotation$FullRunName[selected_batches]
test_matrix <- example_proteome_matrix[, selected_samples]

# with defined color scheme:
color_list <- sample_annotation_to_colors(example_sample_annotation,
    factor_columns = c("MS_batch", "Strain", "Diet", "digestion_batch"),
    numeric_columns = c("DateTime", "order")
)
hierarchical_clustering_plot <- plot_hierarchical_clustering(
    example_proteome_matrix, example_sample_annotation,
    factors_to_plot = c("MS_batch", "Strain", "DateTime", "digestion_batch"),
    color_list = color_list, # can be NULL
    distance = "euclidean", agglomeration = "complete",
    label_samples = FALSE
)
```

---

plot_NA_density *Plot intensity density by missingness*

Description

Compare the distribution of average intensities between features with and without missing observations.

Usage

```
plot_NA_density(x, ...)

## Default S3 method:
plot_NA_density(
  x,
  missing_label = "Missing Value",
  valid_label = "Valid Value",
  palette = c(`Missing Value` = "#A92C23", `Valid Value` = "#345995"),
```

```
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_NA_density(
  x,
  pbf_name = NULL,
  missing_label = "Missing Value",
  valid_label = "Valid Value",
  palette = c(`Missing Value` = "#A92C23", `Valid Value` = "#345995"),
  nrow = NULL,
  ncol = NULL,
  facet_scales = "free_y",
  ...
)
```

Arguments

`x`	A data container. For the ‘ProBatchFeatures‘ method this must be a ‘ProBatch-Features‘ object. The default method accepts any matrix-like input (including ‘SummarizedExperiment‘).
`...`	Additional parameters forwarded to [pheatmap::pheatmap()].
`missing_label`, `valid_label`	Labels used to distinguish rows with and without missing values.
`palette`	Named vector of colours mapped to ‘missing_label‘ and ‘valid_label‘.
`pbf_name`	Character scalar or vector with assay names to plot. When ‘NULL‘, the most recent assay returned by [pb_current_assay()] is used. Only used by the ‘Pro-BatchFeatures‘ method.
`nrow`, `ncol`	Integers controlling the layout when multiple assays are plotted. If both are ‘NULL‘, a roughly square layout is chosen automatically.
`facet_scales`	Scaling behaviour passed to [ggplot2::facet_wrap()] when multiple assays are plotted.

Value

A ‘ggplot‘ object.

`plot_NA_frequency` *Plot missing-value frequency distribution*

Description

Display how many features are observed in a given number of samples.

Usage

```
plot_NA_frequency(x, ...)

## Default S3 method:
plot_NA_frequency(x, show_percent = FALSE, fill = "#345995", ...)

## S3 method for class 'ProBatchFeatures'
plot_NA_frequency(
  x,
  pbf_name = NULL,
  fill = "#345995",
  nrow = NULL,
  ncol = NULL,
  facet_scales = "free_y",
  show_percent = FALSE,
  ...
)
```

Arguments

x	A data container. For the ‘ProBatchFeatures‘ method this must be a ‘ProBatch-Features‘ object. The default method accepts any matrix-like input (including ‘SummarizedExperiment‘).
...	Additional parameters forwarded to [pheatmap::pheatmap()].
show_percent	Logical; display percentages instead of raw counts.
fill	Colour used for the columns in the frequency plot.
pbf_name	Character scalar or vector with assay names to plot. When ‘NULL‘, the most recent assay returned by [pb_current_assay()] is used. Only used by the ‘Pro-BatchFeatures‘ method.
nrow, ncol	Integers controlling the layout when multiple assays are plotted. If both are ‘NULL‘, a roughly square layout is chosen automatically.
facet_scales	Scaling behaviour for facets when plotting multiple assays.

Value

A ‘ggplot‘ object showing the frequency distribution.

---

plot_NA_heatmap *Plot missing-value heatmap(s)*

Description

Functions for visualising the missingness pattern of assay intensities as a binary heatmap. The ‘ProBatchFeatures‘ method supports drawing multiple assays at once by arranging the resulting heatmaps into a user-controlled grid layout.

Usage

```
plot_NA_heatmap(x, ...)

## Default S3 method:
plot_NA_heatmap(
  x,
  sample_annotation = NULL,
  sample_id_col = NULL,
  color_by = NULL,
  label_by = NULL,
  cluster_samples = TRUE,
  cluster_features = TRUE,
  show_row_dend = TRUE,
  show_column_dend = FALSE,
  missing_color = "black",
  valid_color = "grey90",
  col_vector = NULL,
  drop_complete = TRUE,
  draw = TRUE,
  main = NULL,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_NA_heatmap(
  x,
  pbf_name = NULL,
  color_by = NULL,
  label_by = NULL,
  sample_id_col = NULL,
  cluster_samples = TRUE,
  cluster_features = TRUE,
  show_row_dend = TRUE,
  show_column_dend = FALSE,
  missing_color = "black",
  valid_color = "grey90",
  col_vector = NULL,
  drop_complete = TRUE,
  nrow = NULL,
  ncol = NULL,
  draw = TRUE,
  use_subset = TRUE,
  ...
)
```

Arguments

- `x` — A data container. For the ‘ProBatchFeatures‘ method this must be a ‘ProBatch-Features‘ object. The default method accepts any matrix-like input (including ‘SummarizedExperiment‘).
- `...` — Additional parameters forwarded to [pheatmap::pheatmap()].
- `sample_annotation` — Optional data frame with sample-level metadata. Row names (or the column specified via ‘sample_id_col‘) must match the column names of the intensity matrix. When ‘x‘ is a ‘ProBatchFeatures‘ object the sample annotation defaults to ‘as.data.frame(colData(x))‘.
- `sample_id_col` — Optional column in ‘sample_annotation‘ providing unique sample identifiers. Use this when the data frame lacks row names matching the assay column names.
- `color_by` — Optional column name in ‘sample_annotation‘ used to annotate heatmap columns. Use ‘NULL‘ (default) or the string "No" to omit the annotation bar.
- `label_by` — Optional column name (or character vector) used for column labels. Use ‘NULL‘ for default assay column names or the string "No" to suppress column labels entirely.
- `cluster_samples, cluster_features` — Logical flags controlling whether the heatmap columns/rows are clustered.
- `show_row_dend, show_column_dend` — Logical, whether dendrograms should be drawn for the clustered rows/columns.
- `missing_color, valid_color` — Colours used for missing (‘0‘) and observed (‘1‘) values respectively.
- `col_vector` — Optional vector of colours that will be recycled to colour the unique values of ‘color_by‘.
- `drop_complete` — Logical, drop features without any missing values before plotting. Defaults to ‘TRUE‘ to focus on missingness patterns.
- `draw` — Logical, draw the heatmap(s). Set to ‘FALSE‘ to obtain the grob(s) without plotting. For multiple assays, the arranged grob is returned invisibly when ‘draw = TRUE‘.
- `main` — Optional title passed to [pheatmap::pheatmap()] for single assays.
- `pbf_name` — Character scalar or vector with assay names to plot. When ‘NULL‘, the most recent assay returned by [pb_current_assay()] is used. Only used by the ‘ProBatchFeatures‘ method.
- `nrow, ncol` — Integers controlling the layout when multiple assays are plotted. If both are ‘NULL‘, a roughly square layout is chosen automatically.
- `use_subset` — Logical; randomly subset to 5000 rows/columns when assays exceed that size (only used for ‘ProBatchFeatures‘ inputs).

Value

For a single assay the returned value is the ‘pheatmap‘ object. When multiple assays are requested a list is returned invisibly with elements ‘grob‘ (the arranged heatmaps) and ‘heatmaps‘ (individual ‘pheatmap‘ objects). Assays without missing values are skipped with a warning.

plot_PCA *plot PCA plot*

Description

plot PCA plot

Usage

```
## Default S3 method:
plot_PCA(
  data_matrix,
  sample_annotation,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  color_by = "MS_batch",
  shape_by = NULL,
  PC_to_plot = c(1, 2),
  fill_the_missing = -1,
  color_scheme = "brewer",
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme = "classic",
  base_size = 10,
  point_size = 3,
  point_alpha = 0.8,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_PCA(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  plot_title = NULL,
  return_gridExtra = FALSE,
  plot_ncol = NULL,
  ...
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation`
data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`color_by` column name (as in `sample_annotation`) to color by

`shape_by` Optional column used for point shapes in the PCA plot.

`PC_to_plot` principal component numbers for x and y axis

`fill_the_missing`
numeric value determining how missing values should be substituted. If `NULL`, features with missing values are excluded. If `NULL`, features with missing values are excluded.

`color_scheme` a named vector of colors to map to `batch_col`, names corresponding to the levels of the factor. For continuous variables, vector doesn't need to be named.

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

`width` option determining the output image width

`height` option determining the output image height

`units` units: 'cm', 'in' or 'mm'

`plot_title` title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

`theme` ggplot theme, by default `classic`. Can be easily overriden

`base_size` base size of the text in the plot

`point_size` Point size supplied to `ggplot2::geom_point()`.

`point_alpha` Alpha transparency for plotted points.

`...` Additional arguments forwarded to lower-level plotting helpers.

`pbf_name` Assay name(s) used when `data_matrix` is a `ProBatchFeatures`.

`return_gridExtra`
Logical; return arranged grobs instead of a plot list.

`plot_ncol` Number of columns when arranging multiple assay plots.

Value

ggplot scatterplot colored by factor levels of column specified in `factor_to_color`

See Also

autoplot.pca_common, ggplot

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
matrix_test <- na.omit(example_proteome_matrix)[1:50, ]
pca_plot <- plot_PCA(matrix_test, example_sample_annotation,
    color_by = "MS_batch", plot_title = "PCA colored by MS batch"
)
pca_plot <- plot_PCA(matrix_test, example_sample_annotation,
    color_by = "DateTime", plot_title = "PCA colored by DateTime"
)

color_list <- sample_annotation_to_colors(example_sample_annotation,
    factor_columns = c("MS_batch", "digestion_batch"),
    numeric_columns = c("DateTime", "order")
)
pca_plot <- plot_PCA(matrix_test, example_sample_annotation,
    color_by = "DateTime", color_scheme = color_list[["DateTime"]]
)

pca_file <- tempfile("pca_plot", fileext = ".png")
pca_plot <- plot_PCA(matrix_test, example_sample_annotation,
    color_by = "DateTime", plot_title = "PCA colored by DateTime",
    filename = pca_file, width = 14, height = 9, units = "cm"
)
unlink(pca_file)
```

---

plot_peptide_corr_distribution

Create violin plot of peptide correlation distribution

Description

Plot distribution of peptide correlations within one protein and between proteins

Usage

```
plot_peptide_corr_distribution(
  data_matrix,
  peptide_annotation,
  protein_col = "ProteinName",
  feature_id_col = "peptide_group_label",
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
```

```
  plot_title = "Distribution of peptide correlation",
  theme = "classic"
)

plot_peptide_corr_distribution.corrDF(
  corr_distribution,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = "Correlation of peptides",
  theme = "classic",
  base_size = 20
)
```

Arguments

- `data_matrix` — features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)
- `peptide_annotation` — long format data frame with peptide ID and their corresponding protein and/or gene annotations. See `help("example_peptide_annotation")`.
- `protein_col` — column where protein names are specified
- `feature_id_col` — name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.
- `filename` — path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
- `width` — option determining the output image width
- `height` — option determining the output image height
- `units` — units: 'cm', 'in' or 'mm'
- `plot_title` — title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
- `theme` — ggplot theme, by default `classic`. Can be easily overriden
- `corr_distribution` — data frame with peptide correlation distribution
- `base_size` — base font size

Value

`ggplot` object (violin plot of peptide correlation)

See Also

`calculate_peptide_corr_distr`, `ggplot`

Examples

```
data(list = c("example_peptide_annotation", "example_proteome_matrix"), package = "proBatch")
peptide_corr_distribution <- plot_peptide_corr_distribution(
    example_proteome_matrix,
    example_peptide_annotation,
    protein_col = "Gene"
)

data(list = c("example_peptide_annotation", "example_proteome_matrix"), package = "proBatch")
selected_genes <- c("BOVINE_A1ag", "BOVINE_FetuinB", "Cyfip1")
gene_filter <- example_peptide_annotation$Gene %in% selected_genes
peptides_ann <- example_peptide_annotation$peptide_group_label
selected_peptides <- peptides_ann[gene_filter]
matrix_test <- example_proteome_matrix[selected_peptides, ]
pep_annotation_sel <- example_peptide_annotation[gene_filter, ]
corr_distribution <- calculate_peptide_corr_distr(matrix_test,
    pep_annotation_sel,
    protein_col = "Gene"
)
peptide_corr_distribution <- plot_peptide_corr_distribution.corrDF(corr_distribution)

peptide_corr_file <- tempfile("peptide_corr", fileext = ".png")
peptide_corr_distribution <- plot_peptide_corr_distribution.corrDF(corr_distribution,
    filename = peptide_corr_file,
    width = 28, height = 28, units = "cm"
)
unlink(peptide_corr_file)
```

---

plot_protein_corrplot *Peptide correlation matrix (heatmap)*

Description

Plots correlation plot of peptides from a single protein

Usage

```
plot_protein_corrplot(
  data_matrix,
  protein_name,
  peptide_annotation = NULL,
  protein_col = "ProteinName",
  feature_id_col = "peptide_group_label",
  factors_to_plot = c("ProteinName"),
  cluster_rows = FALSE,
  cluster_cols = FALSE,
 heatmap_color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100),
  color_list = NULL,
```

```
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  ...
)
```

Arguments

data_matrix	features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)
protein_name	the name of the protein
peptide_annotation	long format data frame with peptide ID and their corresponding protein and/or gene annotations. See `help("example_peptide_annotation")`.
protein_col	column where protein names are specified
feature_id_col	name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.
factors_to_plot	vector of technical and biological covariates to be plotted in this diagnostic plot (assumed to be present in `sample_annotation`)
cluster_rows	boolean values determining if rows should be clustered or `hclust` object
cluster_cols	boolean values determining if columns should be clustered or `hclust` object
heatmap_color	vector of colors used in heatmap.
color_list	list, as returned by `sample_annotation_to_colors`, where each item contains a color vector for each factor to be mapped to the color.
filename	path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported
width	option determining the output image width
height	option determining the output image height
units	units: 'cm', 'in' or 'mm'
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
...	parameters for the corrplot visualisation

Value

`pheatmap` object

Examples

```
data(list = c("example_peptide_annotation", "example_proteome_matrix"), package = "proBatch")
protein_corrplot_plot <- plot_protein_corrplot(example_proteome_matrix,
    protein_name = "Haao", peptide_annotation = example_peptide_annotation,
    protein_col = "Gene"
)

protein_corrplot_plot <- plot_protein_corrplot(example_proteome_matrix,
    protein_name = c("Haao", "Dhtkd1"),
    peptide_annotation = example_peptide_annotation,
    protein_col = "Gene", factors_to_plot = "Gene"
)
```

---

plot_PVCA *Plot variance distribution by variable*

Description

Plot variance distribution by variable

Usage

```
## Default S3 method:
plot_PVCA(
  data_matrix,
  sample_annotation,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  technical_factors = c("MS_batch", "instrument"),
  biological_factors = c("cell_line", "drug_dose"),
  fill_the_missing = -1,
  pca_threshold = 0.6,
  variance_threshold = 0.01,
  colors_for_bars = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme = "classic",
  base_size = 15,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_PVCA(
```

```
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  plot_title = NULL,
  return_gridExtra = FALSE,
  plot_ncol = NULL,
  ...
)
```

Arguments

`data_matrix` Input object: matrix-like data or a 'ProBatchFeatures' instance.

`sample_annotation`
: data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`technical_factors`
: vector `sample_annotation` column names that are technical covariates

`biological_factors`
: vector `sample_annotation` column names, that are biologically meaningful covariates

`fill_the_missing`
: numeric value determining how missing values should be substituted. If `NULL`, features with missing values are excluded. If `NULL`, features with missing values are excluded.

`pca_threshold` the percentile value of the minimum amount of the variabilities that the selected principal components need to explain

`variance_threshold`
: the percentile value of weight each of the covariates needs to explain (the rest will be lumped together)

`colors_for_bars`
: four-item color vector, specifying colors for the following categories: c('residual', 'biological', 'biol:techn', 'technical')

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

width	option determining the output image width
height	option determining the output image height
units	units: 'cm', 'in' or 'mm'
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
theme	ggplot theme, by default classic. Can be easily overriden
base_size	base size of the text in the plot
...	Additional arguments passed to lower-level methods.
pbf_name	Assay name(s) used when 'data_matrix' is a 'ProBatchFeatures'.
return_gridExtra	Logical; return arranged grobs instead of a plot list.
plot_ncol	Number of columns when arranging multiple assay plots.

Value

ggplot object with the plot

See Also

sample_annotation_to_colors, ggplot

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
matrix_test <- na.omit(example_proteome_matrix)[1:50, ]

pvca_file <- tempfile("pvca", fileext = ".png")
pvca_plot <- plot_PVCA(
    matrix_test,
    example_sample_annotation,
    technical_factors = c("MS_batch", "digestion_batch"),
    biological_factors = c("Diet", "Sex", "Strain"),
    filename = pvca_file, # save to file, can be NULL
    width = 12, height = 8, units = "cm"
)
unlink(pvca_file)
```

---

plot_PVCA.df *plot PVCA, when the analysis is completed*

Description

plot PVCA, when the analysis is completed

Usage

```
## S3 method for class 'df'
plot_PVCA(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  colors_for_bars = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme = "classic",
  base_size = 15,
  return_gridExtra = FALSE,
  plot_ncol = NULL,
  ...
)
```

Arguments

`data_matrix` Data frame of PVCA weights, typically the result of ‘prepare_PVCA_df()‘, or a ‘ProBatchFeatures‘ object.

`pbf_name` Assay name(s) used when ‘data_matrix‘ is a ‘ProBatchFeatures‘.

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix`) are found.

`colors_for_bars` four-item color vector, specifying colors for the following categories: c('residual', 'biological', 'biol:techn', 'technical')

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

`width` option determining the output image width

`height` option determining the output image height

units	units: 'cm', 'in' or 'mm'
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
theme	ggplot theme, by default `classic`. Can be easily overriden
base_size	base size of the text in the plot
return_gridExtra	Logical; return arranged grobs instead of a plot list.
plot_ncol	Number of columns when arranging multiple assay plots.
...	Additional arguments. When 'data_matrix' is a 'ProBatchFeatures', these are forwarded to 'prepare_PVCA_df()'.

Value

`ggplot` object with bars as weights, colored by bio/tech factors

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
matrix_test <- na.omit(example_proteome_matrix)[1:50, ]
pvca_df_res <- prepare_PVCA_df(matrix_test, example_sample_annotation,
    technical_factors = c("MS_batch", "digestion_batch"),
    biological_factors = c("Diet", "Sex", "Strain"),
    pca_threshold = .6, variance_threshold = .01, fill_the_missing = -1
)
colors_for_bars <- c("grey", "green", "blue", "red")
names(colors_for_bars) <- c("residual", "biological", "biol:techn", "technical")

pvca_plot <- plot_PVCA.df(pvca_df_res, colors_for_bars)
```

---

plot_sample_corr_distribution

Create violin plot of sample correlation distribution

Description

Useful to visualize within batch vs within replicate vs non-related sample correlation

Usage

```
plot_sample_corr_distribution(
  data_matrix,
  sample_annotation,
  repeated_samples = NULL,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  biospecimen_id_col = "EarTag",
```

```
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = "Sample correlation distribution",
  plot_param = "batch_replicate",
  theme = "classic"
)

plot_sample_corr_distribution.corrDF(
  corr_distribution,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = "Sample correlation distribution",
  plot_param = "batch_replicate",
  theme = "classic",
  base_size = 20
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation` data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`repeated_samples` if `NULL`, correlation of all samples is plotted

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`batch_col` column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

`biospecimen_id_col` column in `sample_annotation` that captures the biological sample, that (possibly) was profiled several times as technical replicates. Tip: if such ID is absent, but can be defined from several columns, create new `biospecimen_id` column

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

`width` option determining the output image width `height` option determining the output image height

`units` units: 'cm', 'in' or 'mm'

`plot_title` title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

`plot_param` columns, defined in correlation_df, which is output of `calculate_sample_corr_distr`, specifically,

1. `replicate`
2. `batch_the_same`
3. `batch_replicate`
4. `batches`

`theme` ggplot theme, by default `classic`. Can be easily overriden

`corr_distribution` data frame with correlation distribution, as returned by `calculate_sample_corr_distr`

`base_size` base font size

Value

`ggplot` type object with violin plot for each `plot_param`

See Also

`calculate_sample_corr_distr`, `ggplot`

Examples

```
data(list = c("example_sample_annotation", "example_proteome_matrix"), package = "proBatch")
sample_corr_distribution_plot <- plot_sample_corr_distribution(
    example_proteome_matrix,
    example_sample_annotation,
    batch_col = "MS_batch",
    biospecimen_id_col = "EarTag",
    plot_param = "batch_replicate"
)

data(list = c("example_sample_annotation", "example_proteome_matrix"), package = "proBatch")
corr_distribution <- calculate_sample_corr_distr(
    data_matrix = example_proteome_matrix,
    sample_annotation = example_sample_annotation,
    batch_col = "MS_batch", biospecimen_id_col = "EarTag"
)
sample_corr_distribution_plot <- plot_sample_corr_distribution.corrDF(corr_distribution,
    plot_param = "batch_replicate"
)

sample_corr_file <- tempfile("sample_corr", fileext = ".png")
sample_corr_distribution_plot <- plot_sample_corr_distribution.corrDF(corr_distribution,
    plot_param = "batch_replicate",
    filename = sample_corr_file,
    width = 28, height = 28, units = "cm"
```

```
)
unlink(sample_corr_file)
```

---

plot_sample_corr_heatmap
Sample correlation matrix (heatmap)

Description

Plot correlation of selected samples

Usage

```
plot_sample_corr_heatmap(
  data_matrix,
  samples_to_plot = NULL,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  factors_to_plot = NULL,
  cluster_rows = FALSE,
  cluster_cols = FALSE,
 heatmap_color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100),
  color_list = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = sprintf("Correlation matrix of%s samples",
    ifelse(is.null(samples_to_plot), "", " selected")),
  ...
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`samples_to_plot`
string vector of samples in `data_matrix` to be used in the plot

`sample_annotation`
data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")` `sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`factors_to_plot` vector of technical and biological covariates to be plotted in this diagnostic plot (assumed to be present in `sample_annotation`)

`cluster_rows` boolean values determining if rows should be clustered or `hclust` object

`cluster_cols` boolean values determining if columns should be clustered or `hclust` object

`heatmap_color` vector of colors used in heatmap.

`color_list` list, as returned by `sample_annotation_to_colors`, where each item contains a color vector for each factor to be mapped to the color.

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

`width` option determining the output image width

`height` option determining the output image height

`units` units: 'cm', 'in' or 'mm'

`plot_title` title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

`...` parameters for the `pheatmap` visualisation, for details see examples and help to corresponding functions

Value

pheatmap object

See Also

`pheatmap`

Examples

```
data(list = c("example_sample_annotation", "example_proteome_matrix"), package = "proBatch")
specified_samples <- example_sample_annotation$FullRunName[
    which(example_sample_annotation$order %in% 110:115)
]

sample_corr_heatmap <- plot_sample_corr_heatmap(example_proteome_matrix,
    samples_to_plot = specified_samples,
    factors_to_plot = c("MS_batch", "Diet", "DateTime", "digestion_batch"),
    cluster_rows = FALSE, cluster_cols = FALSE,
    annotation_names_col = TRUE, annotation_legend = FALSE,
    show_colnames = FALSE
)

color_list <- sample_annotation_to_colors(example_sample_annotation,
    factor_columns = c(
        "MS_batch", "EarTag", "Strain",
```

```
        "Diet", "digestion_batch", "Sex"
    ),
    numeric_columns = c("DateTime", "order")
)
sample_corr_heatmap_annotated <- plot_sample_corr_heatmap(log_transform_dm(example_proteome_matrix),
    sample_annotation = example_sample_annotation,
    factors_to_plot = c("MS_batch", "Diet", "DateTime", "digestion_batch"),
    cluster_rows = FALSE, cluster_cols = FALSE,
    annotation_names_col = TRUE,
    show_colnames = FALSE, color_list = color_list
)
```

---

plot_sample_mean_or_boxplot

Plot per-sample mean or boxplots for initial assessment

Description

Plot per-sample mean or boxplots (showing median and quantiles). In ordered samples, e.g. consecutive MS runs, order-associated effects are visualised.

Usage

```
## Default S3 method:
plot_sample_mean(
  x,
  sample_annotation,
  sample_id_col = "FullRunName",
  batch_col = "MS_batch",
  color_by_batch = FALSE,
  color_scheme = "brewer",
  order_col = "order",
  vline_color = "grey",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme_name = c("classic", "minimal", "bw", "light", "dark"),
  base_size = 20,
  ylimits = NULL,
  pbf_name = NULL,
  ...
)

## Default S3 method:
```

```
plot_boxplot(
  x,
  sample_annotation,
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  batch_col = "MS_batch",
  color_by_batch = TRUE,
  color_scheme = "brewer",
  order_col = "order",
  facet_col = NULL,
  filename = NULL,
  width = NA,
  height = NA,
  units = c("cm", "in", "mm"),
  plot_title = NULL,
  theme_name = c("classic", "minimal", "bw", "light", "dark"),
  base_size = 20,
  ylimits = NULL,
  outliers = TRUE,
  pbf_name = NULL,
  ...
)

## S3 method for class 'ProBatchFeatures'
plot_sample_mean(x, pbf_name = NULL, plot_title = NULL, ...)

## S3 method for class 'ProBatchFeatures'
plot_boxplot(
  x,
  pbf_name = NULL,
  sample_id_col = NULL,
  plot_title = NULL,
  plot_ncol = NULL,
  return_gridExtra = FALSE,
  ...
)

plot_sample_mean(x, ...)

plot_boxplot(x, ...)
```

Arguments

`x` Input object supplied to the generics (matrix, long data frame, or ‘ProBatchFeatures‘).

`sample_annotation`
data frame with:

1. `sample_id_col` (this can be repeated as row names)

2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`batch_col` column in `sample_annotation` that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

`color_by_batch` (logical) whether to color points and connecting lines by batch factor as defined by `batch_col`.

`color_scheme` named vector, names corresponding to unique batch values of `batch_col` in `sample_annotation`. Best created with sample_annotation_to_colors

`order_col` column in `sample_annotation` that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

`vline_color` color of vertical lines, typically denoting different MS batches in ordered runs; should be `NULL` for experiments without intrinsic order

`facet_col` column in `sample_annotation` with a batch factor to separate plots into facets; usually 2nd to `batch_col`. Most meaningful for multi-instrument MS experiments (where each instrument has its own order-associated effects (see `order_col`) or simultaneous examination of two batch factors (e.g. preparation day and measurement day). For single-instrument case should be set to 'NULL'

`filename` path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

`width` option determining the output image width

`height` option determining the output image height

`units` units: 'cm', 'in' or 'mm'

`plot_title` title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

`theme_name` Name of the ggplot theme to apply to the resulting plot.

`base_size` base font size

`ylimits` range of y-axis to compare two plots side by side, if required.

`pbf_name` Assay name(s) used when 'x' is a 'ProBatchFeatures'.

`...` Additional arguments forwarded between methods.

`measure_col` if `df_long` is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

`outliers` keep (default) or remove the boxplot outliers

`plot_ncol` Number of columns when arranging multiple assay plots.

`return_gridExtra` Logical; return arranged grobs instead of a plot list.

Details

functions for quick visual assessment of trends associated, overall or specific covariate-associated (see `batch_col` and `facet_col`)

Value

ggplot2 class object. Thus, all aesthetics can be overridden

See Also

`ggplot`, date_to_sample_order

Examples

```
data(list = c(
    "example_proteome", "example_sample_annotation",
    "example_proteome_matrix"
), package = "proBatch")

demo_ids <- colnames(example_proteome_matrix)[1:6]
demo_matrix <- example_proteome_matrix[, demo_ids]
demo_annotation <- example_sample_annotation[
    example_sample_annotation$FullRunName %in% demo_ids,
]

plot_sample_mean(
    demo_matrix,
    demo_annotation,
    order_col = "order",
    batch_col = "MS_batch"
)

demo_proteome <- example_proteome[
    example_proteome$FullRunName %in% demo_ids,
]
plot_boxplot(
    demo_proteome,
    sample_annotation = demo_annotation,
    batch_col = "MS_batch"
)
```

`plot_split_violin_with_boxplot` *Plot split violin plot (convenient to compare distribution before and after)*

Description

Plot split violin plot (convenient to compare distribution before and after)

Usage

```
plot_split_violin_with_boxplot(
  df,
  y_col = "y",
  col_for_color = "m",
  col_for_box = "x",
  colors_for_plot = c("#8f1811", "#F8C333"),
  hlineintercept = NULL,
  plot_title = NULL,
  theme = "classic"
)
```

Arguments

df	data.frame with y_col, col_for_color, col_for_box
y_col	value to explore the distribution of
col_for_color	column to use to map to two colors
col_for_box	column to use to do group comparison
colors_for_plot	colors to map to col_for_color
hlineintercept	NULL: no intercept line; non-null: intercept value ...
plot_title	title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))
theme	ggplot theme, by default classic. Can be easily overriden

Value

ggplot object

Examples

```
df <- data.frame(x = rep(c("A", "B"), each = 100), y = rnorm(200), m = rep(c("C", "D"), 100))
plot_split_violin_with_boxplot(df, y_col = "y", col_for_color = "m", col_for_box = "x")
```

---

prepare_PVCA_df *prepare the weights of Principal Variance Components*

Description

prepare the weights of Principal Variance Components

Usage

```
## Default S3 method:
prepare_PVCA_df(
  data_matrix,
  sample_annotation,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  technical_factors = c("MS_batch", "instrument"),
  biological_factors = c("cell_line", "drug_dose"),
  fill_the_missing = -1,
  pca_threshold = 0.6,
  variance_threshold = 0.01,
  ...
)

## S3 method for class 'ProBatchFeatures'
prepare_PVCA_df(
  data_matrix,
  pbf_name = NULL,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  ...
)
```

Arguments

`data_matrix` features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use `help("example_proteome_matrix")`)

`sample_annotation`
data frame with:

1. `sample_id_col` (this can be repeated as row names)
2. biological covariates
3. technical covariates (batches etc)

. See `help("example_sample_annotation")`

`feature_id_col` name of the column with feature/gene/peptide/protein ID used in the long format representation `df_long`. In the wide formatted representation `data_matrix` this corresponds to the row names.

`sample_id_col` name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

`technical_factors`
vector `sample_annotation` column names that are technical covariates

`biological_factors`
vector `sample_annotation` column names, that are biologically meaningful covariates

fill_the_missing
: numeric value determining how missing values should be substituted. If NULL, features with missing values are excluded. If NULL, features with missing values are excluded.

pca_threshold
: the percentile value of the minimum amount of the variabilities that the selected principal components need to explain

variance_threshold
: the percentile value of weight each of the covariates needs to explain (the rest will be lumped together)

...
: Additional arguments forwarded between methods.

pbf_name
: Assay name(s) used when ‘data_matrix‘ is a ‘ProBatchFeatures‘.

Value

data frame with weights and factors, combined in a way ready for plotting

Examples

```
data(list = c("example_proteome_matrix", "example_sample_annotation"), package = "proBatch")
matrix_test <- na.omit(example_proteome_matrix)[1:50, ]

pvca_df_res <- prepare_PVCA_df(matrix_test, example_sample_annotation,
    technical_factors = c("MS_batch", "digestion_batch"),
    biological_factors = c("Diet", "Sex", "Strain"),
    pca_threshold = .6, variance_threshold = .01, fill_the_missing = -1
)
```

---

proBatch — *proBatch: A package for diagnostics and correction of batch effects, primarily in proteomics*

Description

The proBatch package contains functions for analyzing and correcting batch effects (unwanted technical variation) from high-thoughput experiments. Although the package has primarily been developed for mass spectrometry proteomics (DIA/SWATH), it has been designed be applicable to most omic data with minor adaptations. It addresses the following needs:

- prepare the data for analysis
- Visualize batch effects in sample-wide and feature-level;
- Normalize and correct for batch effects.

Arguments

df_long
: data frame where each row is a single feature in a single sample. It minimally has a sample_id_col, a feature_id_col and a measure_col, but usually also an m_score (in OpenSWATH output result file). See help("example_proteome") for more details.

data_matrix
: features (in rows) vs samples (in columns) matrix, with feature IDs in rownames and file/sample names as colnames. See "example_proteome_matrix" for more details (to call the description, use help("example_proteome_matrix"))

sample_annotation
: data frame with:

  1. sample_id_col (this can be repeated as row names)
  2. biological covariates
  3. technical covariates (batches etc)

  . See help("example_sample_annotation")

sample_id_col
: name of the column in sample_annotation table, where the filenames (colnames of the data_matrix are found).

measure_col
: if df_long is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.

feature_id_col
: name of the column with feature/gene/peptide/protein ID used in the long format representation df_long. In the wide formatted representation data_matrix this corresponds to the row names.

batch_col
: column in sample_annotation that should be used for batch comparison (or other, non-batch factor to be mapped to color in plots).

order_col
: column in sample_annotation that determines sample order. It is used for in initial assessment plots (plot_sample_mean_or_boxplot) and feature-level diagnostics (feature_level_diagnostics). Can be 'NULL' if sample order is irrelevant (e.g. in genomic experiments). For more details, order definition/inference, see define_sample_order and date_to_sample_order

facet_col
: column in sample_annotation with a batch factor to separate plots into facets; usually 2nd to batch_col. Most meaningful for multi-instrument MS experiments (where each instrument has its own order-associated effects (see order_col) or simultaneous examination of two batch factors (e.g. preparation day and measurement day). For single-instrument case should be set to 'NULL'

color_by_batch
: (logical) whether to color points and connecting lines by batch factor as defined by batch_col.

peptide_annotation
: long format data frame with peptide ID and their corresponding protein and/or gene annotations. See help("example_peptide_annotation").

color_scheme
: a named vector of colors to map to batch_col, names corresponding to the levels of the factor. For continuous variables, vector doesn't need to be named.

color_list
: list, as returned by sample_annotation_to_colors, where each item contains a color vector for each factor to be mapped to the color.

factors_to_plot
: vector of technical and biological covariates to be plotted in this diagnostic plot (assumed to be present in sample_annotation)

protein_col
: column where protein names are specified

no_fit_imputed
: (logical) whether to use imputed (requant) values, as flagged in qual_col by qual_value for data transformation

qual_col
: column to color point by certain value denoted by qual_value. Design with inferred/requant values in OpenSWATH output data, which means argument value has to be set to m_score.

qual_value
: value in qual_col to color. For OpenSWATH data, this argument value has to be set to 2 (this is an m_score value for imputed values (requant values).

plot_title
: title of the plot (e.g., processing step + representation level (fragments, transitions, proteins) + purpose (meanplot/corrplot etc))

keep_all
: when transforming the data (normalize, correct) - acceptable values: all/default/minimal (which set of columns be kept).

theme
: ggplot theme, by default classic. Can be easily overriden

filename
: path where the results are saved. If null the object is returned to the active window; otherwise, the object is save into the file. Currently only pdf and png format is supported

width
: option determining the output image width

height
: option determining the output image height

units
: units: 'cm', 'in' or 'mm'

base_size
: base font size

Details

To learn more about proBatch, start with the vignettes: browseVignettes(package = "proBatch")

Section

Common arguments to the functions.

Author(s)

Maintainer: Yuliya Burankova <yuliya.burankova@uni-hamburg.de>

Authors:

- Jelena Cuklina <chuklina.jelena@gmail.com>
- Chloe H. Lee <chloe.h.lee94@gmail.com>
- Patrick Pedrioli <pedrioli@gmail.com>
- Olga Zolotareva <olga.zolotareva@uni-hamburg.de>

See Also

Useful links:

- https://github.com/Freddsle/proBatch
- Report bugs at https://github.com/Freddsle/proBatch/issues

Examples

```
if (interactive()) {
    browseVignettes(package = "proBatch")
}
```

ProBatchFeatures — *Construct a ProBatchFeatures object from a wide matrix + sample annotation.*

Description

Construct a ProBatchFeatures object from a wide matrix + sample annotation.

Usage

```
ProBatchFeatures(
  data_matrix,
  sample_annotation = NULL,
  sample_id_col = "FullRunName",
  name = NULL,
  level = "feature"
)
```

Arguments

data_matrix	numeric matrix (features x samples)
sample_annotation	data.frame with sample metadata (rows = samples)
sample_id_col	character(1), column in sample_annotation that matches colnames(data_matrix) If missing, rownames(sample_annotation) are used.
name	character(1), optional; if missing, name is "<level>::raw". If only a single value is provided to the function, without specifying whether it is a name or level, it will be used as the name value.
level	character label like "peptide"/"protein" (default "feature").

Value

A ‘ProBatchFeatures‘ object.

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object --------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
```

```
))

# Long-format constructor ---------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels ---------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`ProBatchFeatures-class` *ProBatchFeatures: QFeatures subclass with operation log, levels/pipelines, and lazy storage*

Description

Assay naming convention: "<level>::<pipeline>" e.g., "peptide::raw", "protein::median_on_log" Pipelines are strings produced by get_chain(as_string=TRUE), e.g., "combat_on_medianNorm_on_log".

Details

Ephemeral "fast" steps are computed but not stored by default (store_fast_steps = FALSE). Use pb_eval() to compute and return data after a step/pipeline without storing. Use pb_transform() to build pipelines and optionally materialize the final assay.

Slots

`chain` character() ordered list of steps (e.g., c("log","medianNorm","combat")).

`oplog` S4Vectors::DataFrame with columns: - step (character), fun (character), from (character), to (character), params (list), timestamp (POSIXct), pkg (character)

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object -------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -----------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format
```

```
# Pipeline evaluation without storing -------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor --------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels -------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

`ProBatchFeatures_from_long`
Construct from LONG df via proBatch::long_to_matrix

Description

Construct from LONG df via proBatch::long_to_matrix

Usage

```
ProBatchFeatures_from_long(
  df_long,
  sample_annotation = NULL,
  feature_id_col = "peptide_group_label",
  sample_id_col = "FullRunName",
  measure_col = "Intensity",
  level = "feature",
  name = NULL
)
```

Arguments

df_long	Data frame in long format with feature/sample/value columns.
sample_annotation	Optional sample metadata aligned to the samples.
feature_id_col	Column containing feature identifiers in ‘df_long‘.
sample_id_col	Column containing sample identifiers in ‘df_long‘.
measure_col	Column with the measured intensity values.
level	Character label describing the biological level of the assay.
name	Optional pipeline name; defaults to ‘<level>::raw‘ when missing.

Value

A ‘ProBatchFeatures‘ object constructed from the long-format input.

Examples

```
# Shared setup for ProBatchFeatures documentation examples -----------------
data("example_ecoli_data", package = "proBatch")

# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)

# Construct a ProBatchFeatures object -------------------------------------
pbf <- ProBatchFeatures(
    data_matrix = all_precursors,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    level = "peptide"
)

# Register a custom step and evaluate it -----------------------------------
```

```
pb_register_step("add_one", function(x) x + 1)
head(pb_eval(pbf, from = "peptide::raw", steps = "add_one"))

# Derived objects for downstream helpers -----------------------------------
pbf_logged <- pb_transform(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm"),
    store_fast_steps = TRUE
)

# Get information about the object ---------------------------------------
get_operation_log(pbf_logged)
get_chain(pbf_logged)
get_chain(pbf_logged, as_string = TRUE)
pb_pipeline_name(pbf_logged) # the latest pipeline
pb_pipeline_name(pbf_logged, assay = "peptide::raw")

# Access assays and matrices -------------------------------------------------
head(pb_current_assay(pbf_logged)) # the latest assay
head(pb_assay_matrix(pbf_logged)) # the latest matrix
head(pb_assay_matrix(pbf_logged, assay = "peptide::raw")) # the latest matrix
head(pb_as_wide(pbf_logged)) # the latest assay in wide format
head(pb_as_long(pbf_logged, sample_id_col = "Run")) # the latest assay in long format

# Pipeline evaluation without storing --------------------------------------
head(pb_eval(
    pbf,
    from = "peptide::raw",
    steps = c("log2", "medianNorm")
))

# Long-format constructor ----------------------------------------------------
long_pbf <- pb_as_long(pbf_logged, sample_id_col = "Run") # the latest assay in long format

ProBatchFeatures_from_long(
    df_long = long_pbf,
    sample_annotation = all_metadata,
    sample_id_col = "Run",
    feature_id_col = "feature_label",
    level = "peptide"
)


# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
```

```
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

# Aggregate and add levels --------------------------------------------------
pb_aggregate_level(
    pbf,
    from = "peptide::raw",
    feature_var = "ProteinID",
    new_level = "protein_new"
)
```

---

sample_annotation_to_colors

Generate colors for sample annotation

Description

Convert the sample annotation data frame to list of colors the list is named as columns included to use in plotting functions

Usage

```
## Default S3 method:
sample_annotation_to_colors(
  sample_annotation,
  sample_id_col = "FullRunName",
  factor_columns = NULL,
  numeric_columns = NULL,
  rare_categories_to_other = TRUE,
  guess_factors = FALSE,
  numeric_palette_type = "brewer",
  ...
)

## S3 method for class 'ProBatchFeatures'
sample_annotation_to_colors(sample_annotation, ...)

sample_annotation_to_colors(sample_annotation, ...)
```

Arguments

`sample_annotation`
: Input object supplied to the generic (data frame or ‘ProBatchFeatures‘).

`sample_id_col`
: name of the column in `sample_annotation` table, where the filenames (colnames of the `data_matrix` are found).

factor_columns
: columns of sample_annotation to be treated as factors. Sometimes categorical variables are depicted as integers (e.g. in column "Batch", values are 1, 2 and 3), specification here allows to map them correctly to qualitative palettes.

numeric_columns
: columns of sample_annotation to be treated as continuous numeric values.

rare_categories_to_other
: if True rare categories will be merged into the value "other"

guess_factors
: whether attempt which of the factor_columns are actually numeric

numeric_palette_type
: palette to be used for numeric values coloring (can be 'brewer' and 'viridis')

...
: Additional arguments forwarded to method implementations.

Value

list of colors for the selected annotation columns. Use color_list_to_df if a data frame representation is needed.

Examples

```
data("example_sample_annotation", package = "proBatch")
color_scheme <- sample_annotation_to_colors(
    example_sample_annotation,
    factor_columns = c(
        "MS_batch", "EarTag", "Strain",
        "Diet", "digestion_batch", "Sex"
    ),
    numeric_columns = c("DateTime", "order")
)
```

---

subset_keep_cols *Subset columns according to 'keep_all' argument*

Description

Helper for data transformation functions to consistently keep a predefined set of columns.

Usage

```
subset_keep_cols(
  df,
  keep_all = "default",
  default_cols = names(df),
  minimal_cols = default_cols
)
```

Arguments

df	data frame to subset
keep_all	one of ‘"all"‘, ‘"default"‘ or ‘"minimal"‘
default_cols	columns to keep when ‘keep_all = "default"‘
minimal_cols	columns to keep when ‘keep_all = "minimal"‘

Value

data frame with selected columns

Examples

```
## Not run:
df <- data.frame(a = 1:3, b = 4:6, c = 7:9)
subset_keep_cols(df, keep_all = "minimal", minimal_cols = c("a", "c"))

## End(Not run)
```

---

transform_raw_data *Functions to log transform raw data before normalization and batch correction*

Description

Functions to log transform raw data before normalization and batch correction

Log transformation of the data long format.

"Unlog" transformation of the data to pre-log form (for quantification, forcing log-transform)

Log transformation of the data matrix format.

Usage

```
log_transform_df(df_long, log_base = 2, offset = 1, measure_col = "Intensity")

unlog_df(df_long, log_base = 2, offset = 1, measure_col = "Intensity")

log_transform_dm(x, ...)

## Default S3 method:
log_transform_dm(x, log_base = 2, offset = 1, ...)

## S3 method for class 'ProBatchFeatures'
log_transform_dm(
  x,
  log_base = 2,
  offset = 1,
```

```
  pbf_name = NULL,
  final_name = NULL,
  ...
)

unlog_dm(x, ...)

## Default S3 method:
unlog_dm(x, log_base = 2, offset = 1, ...)

## S3 method for class 'ProBatchFeatures'
unlog_dm(x, log_base = 2, offset = 1, pbf_name = NULL, final_name = NULL, ...)
```

Arguments

df_long	data frame where each row is a single feature in a single sample. It minimally has a sample_id_col, a feature_id_col and a measure_col, but usually also an m_score (in OpenSWATH output result file). See help("example_proteome") for more details.
log_base	base of the logarithm for transformation
offset	small positive number to prevent 0 conversion to -Inf
measure_col	if df_long is among the parameters, it is the column with expression/abundance/intensity; otherwise, it is used internally for consistency.
x	Input object supplied to the generics (long data frame, matrix, or ‘ProBatchFeatures‘).
...	Additional arguments forwarded between method implementations.
pbf_name	Assay name to transform when ‘x‘ is a ‘ProBatchFeatures‘.
final_name	Optional name for the stored assay produced by the S3 methods.

Value

‘log_transform_df()‘ returns df_long-size data frame, with measure_col log transformed; with old value in another column called "beforeLog_intensity" if "intensity" was the value of measure_col; ‘log_transform_dm()‘ returns data_matrix format matrix

Examples

```
data(list = c("example_proteome", "example_proteome_matrix"), package = "proBatch")

log_transformed_df <- log_transform_df(example_proteome)

log_transformed_matrix <- log_transform_dm(example_proteome_matrix,
    log_base = 10, offset = 1
)
```

warn_unmapped_columns *Warn about unmapped columns*

Description

Emit a warning if some columns will not be mapped to colors.

Usage

```
warn_unmapped_columns(
  sample_annotation,
  columns_for_color_mapping,
  sample_id_col
)
```

Arguments

`sample_annotation`
: data frame containing sample annotations.

`columns_for_color_mapping`
: character vector of columns to be mapped.

`sample_id_col`
: character, the ID column.

Value

No return value, called for side effects (warning).

[,ProBatchFeatures,ANY,ANY,ANY-method *Subset 'ProBatchFeatures' objects without dropping metadata.*

Description

Ensures the '[' method returns a 'ProBatchFeatures' instance so the subclass-specific slots remain available after subsetting.

Usage

```
## S4 method for signature 'ProBatchFeatures,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	A ‘ProBatchFeatures‘ object.
`i`	Row indices passed to the underlying ‘QFeatures‘ subset.
`j`	Column indices passed to the underlying ‘QFeatures‘ subset.
`...`	Additional arguments forwarded to the next method.
`drop`	Logical flag controlling dimension dropping; defaults to ‘TRUE‘.

Value

A ‘ProBatchFeatures‘ object containing the requested subset.

Index