Package ‘metamorphr’

September 1, 2025

Title Tidy and Streamlined Metabolomics Data Workflows

Version 0.1.1

Description Facilitate tasks typically encountered during metabolomics data analysis including data import, filtering, missing value imputation (Stacklies et al. (2007) <doi:10.1093/bioinformatics/btm069>, Stekhoven et al. (2012) <doi:10.1093/bioinformatics/btr597>, Tibshirani et al. (2017) <doi:10.18129/B9.BIOC.IMPUTE>, Troyanskaya et al. (2001) <doi:10.1093/bioinformatics/17.6.520>), normalization (Bolstad et al. (2003) <doi:10.1093/bioinformatics/19.2.185>, Dieterle et al. (2006) <doi:10.1021/ac051632c >, Zhao et al. (2020) <doi:10.1038/s41598-020-72664-6>) transformation, centering and scaling (Van Den Berg et al. (2006) <doi:10.1186/1471-2164-7-142>) as well as statistical tests and plotting. 'metamorphr' introduces a tidy (Wickham et al. (2019) <doi:10.21105/joss.01686>) format for metabolomics data and is designed to make it easier to build elaborate analysis workflows and to integrate them with 'tidyverse' packages including 'dplyr' and 'ggplot2'.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports broom, crayon, dplyr, ggplot2, impute, magrittr, missForest, pcaMethods, purrr, readr, rlang, stats, stringi, tibble, tidyr, utils, vctrs, withr

Depends R (>= 3.5)

LazyData true

Suggests knitr, KODAMA, qsmooth, rmarkdown, stringr, testthat (>= 3.0.0)

Config/testthat/edition 3

URL `https://github.com/yasche/metamorphr`

BugReports `https://github.com/yasche/metamorphr/issues`

VignetteBuilder knitr

NeedsCompilation no

Author Yannik Schermer [aut, cre, cph] (ORCID: <`https://orcid.org/0009-0002-5201-057X`>)

Maintainer Yannik Schermer <`yannik.schermer@chem.rptu.de`>

Repository CRAN

Date/Publication 2025-09-01 17:10:09 UTC

Contents

`calc_neutral_loss` *Calculate neutral losses from precursor ion mass and fragment ion masses*

Description

Calculate neutral loss spectra for all ions with available MSn spectra in `data`. To calculate neutral losses, MSn spectra are required. See `read_mgf`. This step is required for subsequent filtering based on neutral losses (`filter_neutral_loss`). Resulting neutral loss spectra are stored in tibbles in a new list column named `Neutral_Loss`.

Usage

```
calc_neutral_loss(data, m_z_col)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`m_z_col`	Which column holds the precursor m/z? Uses `args_data_masking`.

Value

A tibble with added neutral loss spectra. A new list column is created named `Neutral_Loss`.

Examples

```
toy_mgf %>%
  calc_neutral_loss(m_z_col = PEPMASS)
```

`collapse_max` *Collapse intensities of technical replicates by calculating their maximum*

Description

Calculates the minimum of the intensity of technical replicates (e.g., if the same sample was injected multiple times or if multiple workups have been performed on the same starting material). The function assigns new sample names by joining either group and replicate name, or if a batch column is specified group, replicate and batch together with a specified separator. Due to the nature of the function, sample and feature metadata columns will be dropped unless they are specified with the according arguments.

Usage

```
collapse_max(
  data,
  group_column = .data$Group,
  replicate_column = .data$Replicate,
  batch_column = .data$Batch,
  feature_metadata_cols = "Feature",
  sample_metadata_cols = NULL,
  separator = "_"
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.
`replicate_column`	Which column contains replicate information? Usually `replicate_column = Replicate`. Uses `args_data_masking`.
`batch_column`	Which column contains batch information? If all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`) it will have no effect on the calculation. Usually `batch_column = Batch`. Uses `args_data_masking`.
`feature_metadata_cols`	A character or character vector containing the names of the feature metadata columns. They are usually created when reading the feature table with `read_featuretable`. Feature metadata columns not specified here will be dropped.
`sample_metadata_cols`	A character or character vector containing the names of the sample metadata columns. They are usually created when joining the metadata with `join_metadata`. Sample metadata columns not specified here will be dropped, except for `group_column`, `replicate_column` and `batch_column` if specified.

`separator`	Separator used for joining group and replicate, or group, batch and replicate together to create the new sample names. The new sample names will be Group name, separator, Batch name, separator, Replicate name, or Group name, separator, Replicate name, in case all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`).

Value

A tibble with intensities of technical replicates collapsed.

Examples

```
# uses a slightly modified version of toy_metaboscape_metadata
collapse_toy_metaboscape_metadata <- toy_metaboscape_metadata
collapse_toy_metaboscape_metadata$Replicate <- 1

toy_metaboscape %>%
  join_metadata(collapse_toy_metaboscape_metadata) %>%
  impute_lod() %>%
  collapse_max(group_column = Group, replicate_column = Replicate)
```

`collapse_mean` *Collapse intensities of technical replicates by calculating their mean*

Description

Calculates the mean of the intensity of technical replicates (e.g., if the same sample was injected multiple times or if multiple workups have been performed on the same starting material). The function assigns new sample names by joining either group and replicate name, or if a batch column is specified group, replicate and batch together with a specified separator. Due to the nature of the function, sample and feature metadata columns will be dropped unless they are specified with the according arguments.

Usage

```
collapse_mean(
  data,
  group_column = .data$Group,
  replicate_column = .data$Replicate,
  batch_column = .data$Batch,
  feature_metadata_cols = "Feature",
  sample_metadata_cols = NULL,
  separator = "_"
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.
`replicate_column`	Which column contains replicate information? Usually `replicate_column = Replicate`. Uses `args_data_masking`.
`batch_column`	Which column contains batch information? If all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`) it will have no effect on the calculation. Usually `batch_column = Batch`. Uses `args_data_masking`.
`feature_metadata_cols`	A character or character vector containing the names of the feature metadata columns. They are usually created when reading the feature table with `read_featuretable`. Feature metadata columns not specified here will be dropped.
`sample_metadata_cols`	A character or character vector containing the names of the sample metadata columns. They are usually created when joining the metadata with `join_metadata`. Sample metadata columns not specified here will be dropped, except for `group_column`, `replicate_column` and `batch_column` if specified.
`separator`	Separator used for joining group and replicate, or group, batch and replicate together to create the new sample names. The new sample names will be Group name, separator, Batch name, separator, Replicate name, or Group name, separator, Replicate name, in case all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`).

Value

A tibble with intensities of technical replicates collapsed.

Examples

```
# uses a slightly modified version of toy_metaboscape_metadata
collapse_toy_metaboscape_metadata <- toy_metaboscape_metadata
collapse_toy_metaboscape_metadata$Replicate <- 1

toy_metaboscape %>%
  join_metadata(collapse_toy_metaboscape_metadata) %>%
  impute_lod() %>%
  collapse_mean(group_column = Group, replicate_column = Replicate)
```

`collapse_median` *Collapse intensities of technical replicates by calculating their median*

Description

Calculates the median of the intensity of technical replicates (e.g., if the same sample was injected multiple times or if multiple workups have been performed on the same starting material). The function assigns new sample names by joining either group and replicate name, or if a batch column is specified group, replicate and batch together with a specified separator. Due to the nature of the function, sample and feature metadata columns will be dropped unless they are specified with the according arguments.

Usage

```
collapse_median(
  data,
  group_column = .data$Group,
  replicate_column = .data$Replicate,
  batch_column = .data$Batch,
  feature_metadata_cols = "Feature",
  sample_metadata_cols = NULL,
  separator = "_"
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column` = `Group`. Uses `args_data_masking`.
`replicate_column`	Which column contains replicate information? Usually `replicate_column` = `Replicate`. Uses `args_data_masking`.
`batch_column`	Which column contains batch information? If all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`) it will have no effect on the calculation. Usually `batch_column` = `Batch`. Uses `args_data_masking`.
`feature_metadata_cols`	A character or character vector containing the names of the feature metadata columns. They are usually created when reading the feature table with `read_featuretable`. Feature metadata columns not specified here will be dropped.
`sample_metadata_cols`	A character or character vector containing the names of the sample metadata columns. They are usually created when joining the metadata with `join_metadata`. Sample metadata columns not specified here will be dropped, except for `group_column`, `replicate_column` and `batch_column` if specified.

`separator`	Separator used for joining group and replicate, or group, batch and replicate together to create the new sample names. The new sample names will be Group name, separator, Batch name, separator, Replicate name, or Group name, separator, Replicate name, in case all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`).

Value

A tibble with intensities of technical replicates collapsed.

Examples

```
# uses a slightly modified version of toy_metaboscape_metadata
collapse_toy_metaboscape_metadata <- toy_metaboscape_metadata
collapse_toy_metaboscape_metadata$Replicate <- 1

toy_metaboscape %>%
  join_metadata(collapse_toy_metaboscape_metadata) %>%
  impute_lod() %>%
  collapse_median(group_column = Group, replicate_column = Replicate)
```

`collapse_min` *Collapse intensities of technical replicates by calculating their minimum*

Description

Calculates the minimum of the intensity of technical replicates (e.g., if the same sample was injected multiple times or if multiple workups have been performed on the same starting material). The function assigns new sample names by joining either group and replicate name, or if a batch column is specified group, replicate and batch together with a specified separator. Due to the nature of the function, sample and feature metadata columns will be dropped unless they are specified with the according arguments.

Usage

```
collapse_min(
  data,
  group_column = .data$Group,
  replicate_column = .data$Replicate,
  batch_column = .data$Batch,
  feature_metadata_cols = "Feature",
  sample_metadata_cols = NULL,
  separator = "_"
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.
`replicate_column`	Which column contains replicate information? Usually `replicate_column = Replicate`. Uses `args_data_masking`.
`batch_column`	Which column contains batch information? If all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`) it will have no effect on the calculation. Usually `batch_column = Batch`. Uses `args_data_masking`.
`feature_metadata_cols`	A character or character vector containing the names of the feature metadata columns. They are usually created when reading the feature table with `read_featuretable`. Feature metadata columns not specified here will be dropped.
`sample_metadata_cols`	A character or character vector containing the names of the sample metadata columns. They are usually created when joining the metadata with `join_metadata`. Sample metadata columns not specified here will be dropped, except for `group_column`, `replicate_column` and `batch_column` if specified.
`separator`	Separator used for joining group and replicate, or group, batch and replicate together to create the new sample names. The new sample names will be Group name, separator, Batch name, separator, Replicate name, or Group name, separator, Replicate name, in case all samples belong to the same batch (i.e., they all have the same batch identifier in the `batch_column`).

Value

A tibble with intensities of technical replicates collapsed.

Examples

```
# uses a slightly modified version of toy_metaboscape_metadata
collapse_toy_metaboscape_metadata <- toy_metaboscape_metadata
collapse_toy_metaboscape_metadata$Replicate <- 1

toy_metaboscape %>%
  join_metadata(collapse_toy_metaboscape_metadata) %>%
  impute_lod() %>%
  collapse_min(group_column = Group, replicate_column = Replicate)
```

`create_metadata_skeleton`

Create a blank metadata skeleton

Description

Takes a tidy tibble created by `metamorphr::read_featuretable()` and returns an empty tibble for sample metadata. The tibble can either be populated directly in R or exported and edited by hand (*e.g.* in Excel). Metadata are necessary for several downstream functions. **More columns may be added if necessary**.

Usage

```
create_metadata_skeleton(data)
```

Arguments

`data`	A tidy tibble created by `metamorphr::read_featuretable()`.

Value

An empty tibble structure with the necessary columns for metadata:

Sample The sample name

Group To which group does the samples belong? For example a treatment or a background. Note that additional columns with additional grouping information can be freely added if necessary.

Replicate If multiple technical replicates exist in the data set, they must have the same value for Replicate and the same value for Group so that they can be collapsed. Examples for technical replicates are: the same sample was injected multiple times or workup was performed multiple times with the same starting material. If no technical replicates exist, set `Replicate = 1` for all samples.

Batch The batch in which the samples were prepared or measured. If only one batch exists, set `Batch = 1` for all samples.

Factor A sample-specific factor, for example dry weight or protein content. ...

Examples

```
featuretable_path <- system.file("extdata", "toy_metaboscape.csv", package = "metamorphr")
metadata <- read_featuretable(featuretable_path, metadata_cols = 2:5) %>%
  create_metadata_skeleton()
```

`filter_blank` *Filter Features based on their occurrence in blank samples*

Description

Filters Features based on their occurrence in blank samples. For example, if `min_frac = 3` the maximum intensity in samples must be at least 3 times as high as in blanks for a Feature not to be filtered out.

Usage

```
filter_blank(
  data,
  blank_samples,
  min_frac = 3,
  blank_as_group = FALSE,
  group_column = NULL
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`blank_samples`	Defines the blanks. If `blank_as_group = FALSE` a character vector containing the names of the blank samples as in the `Sample` column of `data`. If `blank_as_group = TRUE` the name(s) of the group(s) that define blanks, as in the `Group` column of `data`. The latter can only be used if sample metadata is provided.
`min_frac`	A numeric defining how many times higher the maximum intensity in samples must be in relation to blanks.
`blank_as_group`	A logical indicating if `blank_samples` are the names of samples or group(s).
`group_column`	Only relevant if `blank_as_group = TRUE`. Which column should be used for grouping blank and non-blank samples? Usually `group_column = Group`. Uses `args_data_masking`.

Value

A filtered tibble.

Examples

```
# Example 1: Define blanks by sample name
toy_metaboscape %>%
  filter_blank(blank_samples = c("Blank1", "Blank2"), blank_as_group = FALSE, min_frac = 3)

# Example 2: Define blanks by group name
# toy_metaboscape %>%
#  join_metadata(toy_metaboscape_metadata) %>%
#  filter_blank(blank_samples = "blank",
```

```
#               blank_as_group = TRUE,
#               min_frac = 3,
#               group_column = Group)
```

`filter_cv` *Filter Features based on their coefficient of variation*

Description

Filters Features based on their coefficient of variation (CV). The CV is defined as $CV = \frac{s_i}{\overline{x_i}}$ with s_i = Standard deviation of sample i and $\overline{x_i}$ = Mean of sample i.

Usage

```
filter_cv(
  data,
  reference_samples,
  max_cv = 0.2,
  ref_as_group = FALSE,
  group_column = NULL,
  na_as_zero = TRUE
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`reference_samples`	The names of the samples or group which will be used to calculate the CV of a feature. Usually Quality Control samples.
`max_cv`	The maximum allowed CV. 0.2 is a reasonable start.
`ref_as_group`	A logical indicating if `reference_samples` are the names of samples or group(s).
`group_column`	Only relevant if `ref_as_group = TRUE`. Which column should be used for grouping reference and non-reference samples? Usually `group_column = Group`. Uses `args_data_masking`.
`na_as_zero`	Should `NA` be replaced with 0 prior to calculation? Under the hood `filter_cv` calculates the CV by `stats::sd(..., na.rm = TRUE) / mean(..., na.rm = TRUE)`. If there are 3 samples to calculate the CV from and 2 of them are `NA` for a specific feature, then the CV for that Feature will be `NA` if `na_as_zero = FALSE`. This might lead to problems. `na_as_zero = TRUE` is the safer pick. Zeros will be replaced with `NA` after calculation no matter if it is `TRUE` or `FALSE`.

Value

A filtered tibble.

References

Coefficient of Variation on Wikipedia

Examples

```
# Example 1: Define reference samples by sample names
toy_metaboscape %>%
  filter_cv(max_cv = 0.2, reference_samples = c("QC1", "QC2", "QC3"))

# Example 2: Define reference samples by group name
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
 filter_cv(max_cv = 0.2, reference_samples = "QC", ref_as_group = TRUE, group_column = Group)
```

`filter_global_mv` *Filter Features based on the absolute number or fraction of samples it was found in*

Description

Filters features based on the number or fraction of samples they are found in. This is usually one of the first steps in metabolomics data analysis and often already performed when the feature table is first created from the raw spectral files..

Usage

```
filter_global_mv(data, min_found = 0.5, fraction = TRUE)
```

Arguments

`data`	A tidy tibble created by `metamorphr::read_featuretable()`.
`min_found`	In how many samples must a Feature be found? If `fraction == TRUE`, a value between 0 and 1 (*e.g.*, 0.5 if a Feature must be found in at least half the samples). If `fraction == FALSE` the absolute maximum number of samples (*e.g.*, 5 if a specific Feature must be found in at least 5 samples).
`fraction`	Either `TRUE` or `FALSE`. Should `min_found` be the absolute number of samples or a fraction?

Value

A filtered tibble.

Examples

```
# Example 1: A feature must be found in at least 50 % of the samples
toy_metaboscape %>%
  filter_global_mv(min_found = 0.5)

# Example 2: A feature must be found in at least 8 samples
toy_metaboscape %>%
  filter_global_mv(min_found = 8, fraction = FALSE)
```

`filter_grouped_mv` *Group-based feature filtering*

Description

Similar to `filter_global_mv` it filters features that are found in a specified number of samples. The key difference is that `filter_grouped_mv()` takes groups into consideration and therefore needs sample metadata. For example, if `fraction` = `TRUE` and `min_found` = `0.5`, a feature must be found in at least 50 % of the samples of at least 1 group. It is very similar to the *Filter features by occurrences in groups* option in Bruker MetaboScape.

Usage

```
filter_grouped_mv(
  data,
  min_found = 0.5,
  group_column = .data$Group,
  fraction = TRUE
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable` with added sample metadata. See ?`create_metadata_skeleton` for help.
`min_found`	Defines in how many samples of at least 1 group a Feature must be found not to be filtered out. If `fraction` == `TRUE`, a value between 0 and 1 (*e.g.*, 0.5 if a Feature must be found in at least half the samples of at least 1 group). If `fraction` == `FALSE` the absolute maximum number of samples (*e.g.*, 5 if a specific Feature must be found in at least 5 samples of at least 1 group).
`group_column`	Which column should be used for grouping? Usually `group_column` = `Group`. Uses `args_data_masking`.
`fraction`	Either `TRUE` or `FALSE`. Should `min_found` be the absolute number of samples or a fraction?

Value

A filtered tibble.

Examples

```
# A Feature must be found in all samples of at least 1 group.
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
  filter_grouped_mv(min_found = 1, group_column = Group)
```

`filter_msn` *Filter Features based on occurrence of fragment ions*

Description

Filters Features based on the presence of MSn fragments. This can help, for example with the identification of potential homologous molecules.

Usage

```
filter_msn(
  data,
  fragments,
  min_found,
  tolerance = 5,
  tolerance_type = "ppm",
  show_progress = TRUE
)
```

Arguments

`data`	A data frame containing MSn spectra.
`fragments`	A numeric. Exact mass of the fragment(s) to filter by.
`min_found`	How many of the `fragments` must be found in order to keep the row? If `min_found = length(fragments)`, all fragments must be found.
`tolerance`	A numeric. The tolerance to apply to the fragments. Either an absolute value in Da (if `tolerance_type = "absolute"`) or in ppm (if `tolerance_type = "ppm"`).
`tolerance_type`	Either `"absolute"` or `"ppm"`. Should the tolerance be an absolute value or in ppm?
`show_progress`	A `logical` indicating whether the progress of the filtering should be printed to the console. Only important for large tibbles.

Value

A filtered tibble.

Examples

```
# all of the given fragments (3) must be found
# returns the first row of toy_mgf
toy_mgf %>%
  filter_msn(fragments = c(12.3456, 23.4567, 34.5678), min_found = 3)

# all of the given fragments (3) must be found
# returns an empty tibble because the third fragment
# of row 1 (34.5678)
# is outside of the tolerance (5 ppm):
# Lower bound:
# 34.5688 - 34.5688 * 5 / 1000000 = 34.5686
# Upper bound:
# 34.5688 + 34.5688 * 5 / 1000000 = 34.5690
toy_mgf %>%
  filter_msn(fragments = c(12.3456, 23.4567, 34.5688), min_found = 3)

# only 2 of the 3 fragments must be found
# returns the first row of toy_mgf
toy_mgf %>%
  filter_msn(fragments = c(12.3456, 23.4567, 34.5688), min_found = 2)
```

`filter_mz` *Filter Features based on their mass-to-charge ratios*

Description

Facilitates filtering by given mass-to-charge ratios (m/z) with a defined tolerance. Can also be used to filter based on exact mass.

Usage

```
filter_mz(data, m_z_col, masses, tolerance = 5, tolerance_type = "ppm")
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`m_z_col`	Which column holds the precursor m/z (or exact mass)? Uses `args_data_masking`.
`masses`	The mass(es) to filter by.
`tolerance`	A numeric. The tolerance to apply to the masses Either an absolute value in Da (if `tolerance_type = "absolute"`) or in ppm (if `tolerance_type = "ppm"`).
`tolerance_type`	Either `"absolute"` or `"ppm"`. Should the tolerance be an absolute value or in ppm?

Value

A filtered tibble.

Examples

```
# Use a tolerance of plus or minus 5 ppm
toy_metaboscape %>%
  filter_mz(m_z_col = `m/z`, 162.1132, tolerance = 5, tolerance_type = "ppm")

# Use a tolerance of plus or minus 0.005 Da
toy_metaboscape %>%
  filter_mz(m_z_col = `m/z`, 162.1132, tolerance = 0.005, tolerance_type = "absolute")
```

`filter_neutral_loss` *Filter Features based on occurrence of neutral losses*

Description

The occurrence of characteristic neutral losses can help with the putative annotation of molecules. See the Reference section for an example.

Usage

```
filter_neutral_loss(
  data,
  losses,
  min_found,
  tolerance = 10,
  tolerance_type = "ppm",
  show_progress = TRUE
)
```

Arguments

`data`	A data frame containing MSn spectra.
`losses`	A numeric. Exact mass of the fragment(s) to filter by.
`min_found`	How many of the `fragments` must be found in order to keep the row? If `min_found = length(fragments)`, all fragments must be found.
`tolerance`	A numeric. The tolerance to apply to the fragments. Either an absolute value in Da (if `tolerance_type = "absolute"`) or in ppm (if `tolerance_type = "ppm"`).
`tolerance_type`	Either `"absolute"` or `"ppm"`. Should the tolerance be an absolute value or in ppm?
`show_progress`	A `logical` indicating whether the progress of the filtering should be printed to the console. Only important for large tibbles.

Value

A filtered tibble.

References

- A. Brink, F. Fontaine, M. Marschmann, B. Steinhuber, E. N. Cece, I. Zamora, A. Pähler, *Rapid Commun. Mass Spectrom.* **2014**, *28*, 2695–2703, DOI 10.1002/rcm.7062.

Examples

```
# neutral losses must be calculated first
toy_mgf_nl <- toy_mgf %>%
  calc_neutral_loss(m_z_col = PEPMASS)

# all of the given losses (3) must be found
# returns the first row of toy_mgf
toy_mgf_nl %>%
  filter_neutral_loss(losses = c(11.1111, 22.2222, 33.3333), min_found = 3)

# all of the given fragments (3) must be found
# returns an empty tibble because the third loss
# of row 1 (33.3333)
# is outside of the tolerance (10 ppm):
# Lower bound:
# 33.4333 - 33.4333 * 5 / 1000000 = 33.4333
# Upper bound:
# 33.4333 + 33.4333 * 5 / 1000000 = 33.4336
toy_mgf_nl %>%
  filter_neutral_loss(losses = c(11.1111, 22.2222, 33.4333), min_found = 3)

# only 2 of the 3 fragments must be found
# returns the first row of toy_mgf
toy_mgf_nl %>%
  filter_neutral_loss(losses = c(11.1111, 22.2222, 33.4333), min_found = 2)
```

`impute_bpca` *Impute missing values using Bayesian PCA*

Description

One of several PCA-based imputation methods. Basically a wrapper around `pcaMethods::pca(method = "bpca")`. For a detailed discussion, see the `vignette("pcaMethods")` and `vignette("missingValues", "pcaMethods")` as well as the References section.

Important Note

`impute_bpca()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_bpca()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `impute_bpca()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data.

Usage

```
impute_bpca(data, n_pcs = 2, center = TRUE, scale = "none", direction = 2)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_pcs`	The number of PCs to calculate.
`center`	Should `data` be mean centered? See `prep` for details.
`scale`	Should `data` be scaled? See `prep` for details.
`direction`	Either `1` or `2`. `1` runs a PCA on a matrix with samples in columns and features in rows and `2` runs a PCA on a matrix with features in columns and samples in rows. Both are valid according to this discussion on GitHub but give **different results**.

Value

A tibble with imputed missing values.

References

- H. R. Wolfram Stacklies, **2017**, DOI 10.18129/B9.BIOC.PCAMETHODS.
- W. Stacklies, H. Redestig, M. Scholz, D. Walther, J. Selbig, *Bioinformatics* **2007**, *23*, 1164–1167, DOI 10.1093/bioinformatics/btm069.

Examples

```
toy_metaboscape %>%
  impute_bpca()
```

`impute_global_lowest` *Impute missing values by replacing them with the lowest observed intensity (global)*

Description

Replace missing intensity values (`NA`) with the lowest observed intensity.

Usage

```
impute_global_lowest(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with imputed missing values.

Examples

```
toy_metaboscape %>%
  impute_global_lowest()
```

`impute_knn` *Impute missing values using nearest neighbor averaging*

Description

Basically a wrapper function around `impute::impute.knn`. Imputes missing values using the k-th nearest neighbor algorithm.

Note that the function ln-transforms the data prior to imputation and transforms it back to the original scale afterwards. **Please do not do it manually prior to calling** `impute_knn()`**!** See References for more information.

Important Note

`impute_knn()` depends on the `impute` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_knn()` is called without the `impute` package installed, you should be asked if you want to install `pak` and `impute`. If you want to use `impute_knn()` you have to install those. In case you run into trouble with the automatic installation, please install `impute` manually. See impute: Imputation for microarray data for instructions on manual installation.

Usage

```
impute_knn(data, quietly = TRUE, ...)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`quietly`	`TRUE` or `FALSE`. Should messages and warnings from `impute.knn` be printed to the console?
`...`	Additional parameters passed to `impute.knn`.

Value

A tibble with imputed missing values.

References

- Robert Tibshirani, Trevor Hastie, **2017**, DOI 10.18129/B9.BIOC.IMPUTE.
- J. Khan, J. S. Wei, M. Ringnér, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, P. S. Meltzer, *Nat Med* **2001**, *7*, 673–679, DOI 10.1038/89044.

Examples

```
toy_metaboscape %>%
  impute_knn()
```

`impute_lls` *Impute missing values using Local Least Squares (LLS)*

Description

Basically a wrapper around `pcaMethods::llsImpute`. For a detailed discussion, see the `vignette("pcaMethods")` and `vignette("missingValues", "pcaMethods")` as well as the References section.

Important Note `impute_lls()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_svd()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `impute_lls()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data for instructions on manual installation.

Usage

```
impute_lls(
  data,
  correlation = "pearson",
  complete_genes = FALSE,
  center = FALSE,
  cluster_size = 10
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`correlation`	The method used to calculate correlations between features. One of `"pearson"`, `"spearman"` or `"kendall"`. See `cor`.
`complete_genes`	If `TRUE` only complete features will be used for regression, if `FALSE`, all will be used.
`center`	Should `data` be mean centered? See `prep` for details.
`cluster_size`	The number of similar features used for regression.

Value

A tibble with imputed missing values.

References

- H. R. Wolfram Stacklies, **2017**, DOI 10.18129/B9.BIOC.PCAMETHODS.
- W. Stacklies, H. Redestig, M. Scholz, D. Walther, J. Selbig, *Bioinformatics* **2007**, *23*, 1164–1167, DOI 10.1093/bioinformatics/btm069.

Examples

```
# The cluster size must be reduced because
# the data set is too small for the default (10)

toy_metaboscape %>%
  impute_lls(complete_genes = TRUE, cluster_size = 5)
```

`impute_lod` *Impute missing values by replacing them with the Feature 'Limit of Detection'*

Description

Replace missing intensity values (`NA`) by what is assumed to be the detector limit of detection (LoD). It is estimated by dividing the Feature minimum by the provided denominator, usually 5. See the References section for more information.

Usage

```
impute_lod(data, div_by = 5)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`div_by`	A numeric value that specifies by which number the Feature minimum will be divided

Value

A tibble with imputed missing values.

References

LoD on OmicsForum

Examples

```
toy_metaboscape %>%
  impute_lod()
```

`impute_mean` *Impute missing values by replacing them with the Feature mean*

Description

Replace missing intensity values (`NA`) with the Feature mean of non-`NA` values. For example, if a Feature has the measured intensities `NA, 1, NA, 3, 2` in samples 1-5, the intensities after `impute_mean()` would be `2, 1, 2, 3, 2`.

Usage

```
impute_mean(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with imputed missing values.

Examples

```
toy_metaboscape %>%
  impute_mean()
```

`impute_median` *Impute missing values by replacing them with the Feature median*

Description

Replace missing intensity values (`NA`) with the Feature median of non-`NA` values. For example, if a Feature has the measured intensities `NA, 1, NA, 3, 2` in samples 1-5, the intensities after `impute_median()` would be `2, 1, 2, 3, 2`.

Usage

```
impute_median(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with imputed missing values.

Examples

```
toy_metaboscape %>%
  impute_median()
```

`impute_min` *Impute missing values by replacing them with the Feature minimum*

Description

Replace missing intensity values (`NA`) with the Feature minimum of non-`NA` values.

Usage

```
impute_min(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with imputed missing values.

Examples

```
toy_metaboscape %>%
  impute_min()
```

`impute_nipals` *Impute missing values using NIPALS PCA*

Description

One of several PCA-based imputation methods. Basically a wrapper around `pcaMethods::pca(method = "nipals")`. For a detailed discussion, see the `vignette("pcaMethods")` and `vignette("missingValues", "pcaMethods")` as well as the References section.

Important Note

`impute_nipals()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_nipals()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `impute_nipals()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data for instructions on manual installation.

Usage

```
impute_nipals(data, n_pcs = 2, center = TRUE, scale = "none", direction = 2)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_pcs`	The number of PCs to calculate.
`center`	Should `data` be mean centered? See `prep` for details.
`scale`	Should `data` be scaled? See `prep` for details.
`direction`	Either `1` or `2`. `1` runs a PCA on a matrix with samples in columns and features in rows and `2` runs a PCA on a matrix with features in columns and samples in rows. Both are valid according to this discussion on GitHub but give **different results**.

Value

A tibble with imputed missing values.

References

- H. R. Wolfram Stacklies, **2017**, DOI 10.18129/B9.BIOC.PCAMETHODS.
- W. Stacklies, H. Redestig, M. Scholz, D. Walther, J. Selbig, *Bioinformatics* **2007**, *23*, 1164–1167, DOI 10.1093/bioinformatics/btm069.

Examples

```
toy_metaboscape %>%
  impute_nipals()
```

`impute_ppca` *Impute missing values using Probabilistic PCA*

Description

One of several PCA-based imputation methods. Basically a wrapper around `pcaMethods::pca(method = "ppca")`. For a detailed discussion, see the `vignette("pcaMethods")` and `vignette("missingValues", "pcaMethods")` as well as the References section. In the underlying function (`pcaMethods::pca(method = "ppca")`), the order of columns has an influence on the outcome. Therefore, calling `pcaMethods::pca(method = "ppca")` on a matrix and calling `metamorphr::impute()` on a tidy tibble might give different results, even though they contain the same data. That is because under the hood, the tibble is transformed to a matrix prior to calling `pcaMethods::pca(method = "ppca")` and you have limited influence on the column order of the resulting matrix.

Important Note

`impute_ppca()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_ppca()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `impute_ppca()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data for instructions on manual installation.

Usage

```
impute_ppca(
  data,
  n_pcs = 2,
  center = TRUE,
  scale = "none",
  direction = 2,
  random_seed = 1L
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_pcs`	The number of PCs to calculate.
`center`	Should `data` be mean centered? See `prep` for details.
`scale`	Should `data` be scaled? See `prep` for details.
`direction`	Either `1` or `2`. `1` runs a PCA on a matrix with samples in columns and features in rows and `2` runs a PCA on a matrix with features in columns and samples in rows. Both are valid according to this discussion on GitHub but give **different results**.
`random_seed`	An integer used as seed for the random number generator.

Value

A tibble with imputed missing values.

References

- H. R. Wolfram Stacklies, **2017**, DOI 10.18129/B9.BIOC.PCAMETHODS.
- W. Stacklies, H. Redestig, M. Scholz, D. Walther, J. Selbig, *Bioinformatics* **2007**, *23*, 1164–1167, DOI 10.1093/bioinformatics/btm069.

Examples

```
toy_metaboscape %>%
  impute_ppca()
```

`impute_rf` *Impute missing values using random forest*

Description

Basically a wrapper function around `missForest::missForest`. Imputes missing values using the random forest algorithm.

Usage

```
impute_rf(data, random_seed = 1L, ...)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`random_seed`	A seed for the random number generator. Can be an integer or `NULL` (in case no particular seed should be used) but for reproducibility reasons it is **strongly advised** to provide an integer.
`...`	Additional parameters passed to `missForest`.

Value

A tibble with imputed missing values.

References

- **missForest** on CRAN
- D. J. Stekhoven, P. Bühlmann, *Bioinformatics* **2012**, *28*, 112–118, DOI 10.1093/bioinformatics/btr597.

Examples

```
toy_metaboscape %>%
  impute_rf()
```

`impute_svd` *Impute missing values using Singular Value Decomposition (SVD)*

Description

Basically a wrapper around `pcaMethods::pca(method = "svdImpute")`. For a detailed discussion, see the `vignette("pcaMethods")` and `vignette("missingValues", "pcaMethods")` as well as the References section.

Important Note `impute_svd()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `impute_svd()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `impute_svd()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data for instructions on manual installation.

Usage

```
impute_svd(data, n_pcs = 2, center = TRUE, scale = "none", direction = 2)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_pcs`	The number of PCs to calculate.
`center`	Should `data` be mean centered? See `prep` for details.
`scale`	Should `data` be scaled? See `prep` for details.
`direction`	Either `1` or `2`. `1` runs `pcaMethods::pca(method = "svdImpute")` on a matrix with samples in columns and features in rows and `2` runs `pcaMethods::pca(method = "svdImpute")` on a matrix with features in columns and samples in rows. Both are valid according to this discussion on GitHub but give **different results**.

Value

A tibble with imputed missing values.

References

- H. R. Wolfram Stacklies, **2017**, DOI 10.18129/B9.BIOC.PCAMETHODS.
- W. Stacklies, H. Redestig, M. Scholz, D. Walther, J. Selbig, *Bioinformatics* **2007**, *23*, 1164–1167, DOI 10.1093/bioinformatics/btm069.
- O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, R. B. Altman, *Bioinformatics* **2001**, *17*, 520–525, DOI 10.1093/bioinformatics/17.6.520.

Examples

```
toy_metaboscape %>%
  impute_svd()
```

`impute_user_value` *Impute missing values by replacing them with a user-provided value*

Description

Replace missing intensity values (`NA`) with a user-provided value (e.g., 1).

Usage

```
impute_user_value(data, value)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`value`	Numeric that replaces missing values

Value

A tibble with imputed missing values.

Examples

```
toy_metaboscape %>%
  impute_user_value(value = 1)
```

`join_metadata` *Join a featuretable and sample metadata*

Description

Joins a featuretable and associated sample metadata. Basically a wrapper around `left_join` where `by = "Sample"`.

Usage

```
join_metadata(data, metadata)
```

Arguments

`data`	A feature table created with `read_featuretable`
`metadata`	Sample metadata created with `create_metadata_skeleton`

Value

A tibble with added sample metadata.

Examples

```
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata)
```

---

normalize_cyclic_loess

Normalize intensities across samples using cyclic LOESS normalization

Description

The steps the algorithm takes are the following:

1. log2 transform the intensities
2. Choose 2 samples to generate an MA-plot from
3. Fit a LOESS curve
4. Subtract half of the difference between the predicted value and the true value from the intensity of sample 1 and add the same amount to the intensity of Sample 2
5. Repeat for all unique combinations of samples
6. Repeat all steps until the model converges or `n_iter` is reached.

Convergence is assumed if the confidence intervals of all LOESS smooths include the 0 line. If `fixed_iter = TRUE`, the algorithm will perform exactly `n_iter` iterations. If `fixed_iter = FALSE`, the algorithm will perform a maximum of `n_iter` iterations.

See the reference section for details.

Usage

```
normalize_cyclic_loess(
  data,
  n_iter = 3,
  fixed_iter = TRUE,
  loess_span = 0.7,
  level = 0.95,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_iter`	The number of iterations to perform. If `fixed_iter = TRUE` exactly `n_iter` will be performed. If `fixed_iter = FALSE` a maximum of `n_iter` will be performed and the algorithm will stop whether convergence is reached or not.
`fixed_iter`	Should a fixed number of iterations be performed?

`loess_span`	The span of the LOESS fit. A larger span produces a smoother line.
`level`	The confidence level for the convergence criterion. Note that a a larger confidence level produces larger confidence intervals and therefore the algorithm stops earlier.
`verbose`	`TRUE` or `FALSE`. Should messages be printed to the console?
`...`	Arguments passed onto `loess`. For example, `degree = 1, family = "symmetric", iterations = 4, s` produces a LOWESS fit.

Value

A tibble with intensities normalized across samples.

References

- B. M. Bolstad, R. A. Irizarry, M. Åstrand, T. P. Speed, *Bioinformatics* **2003**, *19*, 185–193, DOI 10.1093/bioinformatics/19.2.185.
- Karla Ballman, Diane Grill, Ann Oberg, Terry Therneau, “Faster cyclic loess: normalizing DNA arrays via linear models” can be found under https://www.mayo.edu/research/documents/biostat-68pdf/doc-10027897, 2004.
- K. V. Ballman, D. E. Grill, A. L. Oberg, T. M. Therneau, *Bioinformatics* **2004**, *20*, 2778–2786, DOI 10.1093/bioinformatics/bth327.

Examples

```
toy_metaboscape %>%
  impute_lod() %>%
  normalize_cyclic_loess()
```

`normalize_factor` *Normalize intensities across samples using a normalization factor*

Description

Normalization is done by dividing the intensity by a sample-specific factor (e.g., weight, protein or DNA content). This function requires a sample-specific factor, usually supplied via the `Factor` column from the sample metadata. See the Examples section for details.

Usage

```
normalize_factor(data, factor_column = .data$Factor)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`factor_column`	Which column contains the sample-specific factor? Usually `factor_column = Factor`. Uses `args_data_masking`.

Value

A tibble with intensities normalized across samples.

Examples

```
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
  normalize_factor()
```

`normalize_median` *Normalize intensities across samples by dividing by the sample median*

Description

Normalize across samples by dividing feature intensities by the sample median, making the median 1 in all samples. See References for more information.

Usage

```
normalize_median(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with intensities normalized across samples.

References

T. Ramirez, A. Strigun, A. Verlohner, H.-A. Huener, E. Peter, M. Herold, N. Bordag, W. Mellert, T. Walk, M. Spitzer, X. Jiang, S. Sperber, T. Hofmann, T. Hartung, H. Kamp, B. Van Ravenzwaay, *Arch Toxicol* **2018**, *92*, 893–906, DOI 10.1007/s00204-017-2079-6.

Examples

```
toy_metaboscape %>%
  normalize_median()
```

`normalize_pqn` *Normalize intensities across samples using a Probabilistic Quotient Normalization (PQN)*

Description

This method was originally developed for H-NMR spectra of complex biofluids but has been adapted for other 'omics data. It aims to eliminate dilution effects by calculating the most probable dilution factor for each sample, relative to one or more reference samples. See references for more details.

Usage

```
normalize_pqn(
  data,
  fn = "median",
  normalize_sum = TRUE,
  reference_samples = NULL,
  ref_as_group = FALSE,
  group_column = NULL
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`fn`	Which function should be used to calculate the reference spectrum from the reference samples? Can be either "mean" or "median".
`normalize_sum`	A logical indicating whether a sum normalization (aka total area normalization) should be performed prior to PQN. It is recommended to do so and other packages (e.g., **KODAMA**) also perform a sum normalization prior to PQN.
`reference_samples`	Either `NULL` or a character or character vector containing the sample(s) to calculate the reference spectrum from. In the original publication, it is advised to calculate the median of control samples. If `NULL`, all samples will be used to calculate the reference spectrum.
`ref_as_group`	A logical indicating if `reference_samples` are the names of samples or group(s).
`group_column`	Only relevant if `ref_as_group = TRUE`. Which column should be used for grouping reference and non-reference samples? Usually `group_column = Group`. Uses `args_data_masking`.

Value

A tibble with intensities normalized across samples.

References

- F. Dieterle, A. Ross, G. Schlotterbeck, H. Senn, *Anal. Chem.* **2006**, *78*, 4281–4290, DOI 10.1021/ac051632c.

Examples

```
# specify the reference samples with their sample names
toy_metaboscape %>%
  impute_lod() %>%
  normalize_pqn(reference_samples = c("QC1", "QC2", "QC3"))

# specify the reference samples with their group names
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
  impute_lod() %>%
  normalize_pqn(reference_samples = c("QC"), ref_as_group = TRUE, group_column = Group)
```

`normalize_quantile_all` *Normalize intensities across samples using standard Quantile Normalization*

Description

This is the standard approach for Quantile Normalization. Other sub-flavors are also available:

- `normalize_quantile_group`
- `normalize_quantile_batch`
- `normalize_quantile_smooth`

See References for more information.

Usage

```
normalize_quantile_all(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with intensities normalized across samples.

References

Y. Zhao, L. Wong, W. W. B. Goh, *Sci Rep* **2020**, *10*, 15534, DOI 10.1038/s41598-020-72664-6.

Examples

```
toy_metaboscape %>%
  normalize_quantile_all()
```

`normalize_quantile_batch`

Normalize intensities across samples using grouped Quantile Normalization with multiple batches

---

Description

This function performs a Quantile Normalization on each sub-group and batch in the data set. **It therefore requires grouping information**. See Examples for more information. This approach might perform better than the standard approach, `normalize_quantile_all`, if sub-groups are very different (e.g., when comparing cancer vs. normal tissue).

Other sub-flavors are also available:

- `normalize_quantile_all`
- `normalize_quantile_batch`
- `normalize_quantile_smooth`

See References for more information. Note that it is equivalent to the 'Discrete' normalization in Zhao *et al.* but has been renamed for internal consistency.

Usage

```
normalize_quantile_batch(
  data,
  group_column = .data$Group,
  batch_column = .data$Batch
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.
`batch_column`	Which column contains the batch information? Usually `grouping_column = Batch`. Uses `args_data_masking`.

Value

A tibble with intensities normalized across samples.

References

Y. Zhao, L. Wong, W. W. B. Goh, *Sci Rep* **2020**, *10*, 15534, DOI 10.1038/s41598-020-72664-6.

Examples

```
toy_metaboscape %>%
  # Metadata, including grouping and batch information,
  # must be added before using normalize_quantile_batch()
  join_metadata(toy_metaboscape_metadata) %>%
  normalize_quantile_batch(group_column = Group, batch_column = Batch)
```

`normalize_quantile_group` *Normalize intensities across samples using grouped Quantile Normalization*

Description

This function performs a Quantile Normalization on each sub-group in the data set. **It therefore requires grouping information**. See Examples for more information. This approach might perform better than the standard approach, `normalize_quantile_all`, if sub-groups are very different (e.g., when comparing cancer vs. normal tissue).

Other sub-flavors are also available:

- `normalize_quantile_all`
- `normalize_quantile_batch`
- `normalize_quantile_smooth`

See References for more information. Note that it is equivalent to the 'Class-specific' normalization in Zhao *et al.* but has been renamed for internal consistency.

Usage

```
normalize_quantile_group(data, group_column = .data$Group)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.

Value

A tibble with intensities normalized across samples.

References

Y. Zhao, L. Wong, W. W. B. Goh, *Sci Rep* **2020**, *10*, 15534, DOI 10.1038/s41598-020-72664-6.

Examples

```
toy_metaboscape %>%
  # Metadata, including grouping information, must be added before using normalize_quantile_group()
   join_metadata(toy_metaboscape_metadata) %>%
   normalize_quantile_group(group_column = Group)
```

`normalize_quantile_smooth`

Normalize intensities across samples using smooth Quantile Normalization (qsmooth)

Description

This function performs a smooth Quantile Normalization on each sub-group in the data set (qsmooth). **It therefore requires grouping information**. See Examples for more information. This approach might perform better than the standard approach, `normalize_quantile_all`, if sub-groups are very different (e.g., when comparing cancer vs. normal tissue). The result lies somewhere between `normalize_quantile_group` and `normalize_quantile_all`. Basically a re-implementation of Hicks *et al.* (2018).

Usage

```
normalize_quantile_smooth(
  data,
  group_column = .data$Group,
  rolling_window = 0.05
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.
`rolling_window`	`normalize_quantile_smooth` uses a rolling window median to eliminate isolated outliers. This argument specifies the size of the rolling window as a fraction of the number of unique features in `data`. For example, if there are 100 features in `data` and `rolling_window = 0.05`, the rolling median will be calculated from 5 features. Set `rolling_window = 0` to disable.

Value

A tibble with intensities normalized across samples.

References

- S. C. Hicks, K. Okrah, J. N. Paulson, J. Quackenbush, R. A. Irizarry, H. C. Bravo, *Biostatistics* **2018**, *19*, 185–198, DOI 10.1093/biostatistics/kxx028.
- Y. Zhao, L. Wong, W. W. B. Goh, *Sci Rep* **2020**, *10*, 15534, DOI 10.1038/s41598-020-72664-6.

Examples

```
toy_metaboscape %>%
  # Metadata, including grouping information, must be added before using normalize_quantile_group()
  join_metadata(toy_metaboscape_metadata) %>%
  normalize_quantile_smooth(group_column = Group)
```

`normalize_ref` *Normalize intensities across samples using a reference feature*

Description

Performs a normalization based on a reference feature, for example an internal standard. Divides the Intensities of all features by the Intensity of the reference feature in that sample and multiplies them with a constant value, making the Intensity of the reference feature the same in each sample.

Usage

```
normalize_ref(
  data,
  reference_feature,
  identifier_column,
  reference_feature_intensity = 1
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`reference_feature`	An identifier for the reference feature. Must be unique. It is recommended to use the UID.
`identifier_column`	The column in which to look for the reference feature. It is recommended to use `identifier_column = UID`
`reference_feature_intensity`	Either a constant value with which the intensity of each feature is multiplied or a function (e.g., mean, median, min, max). If a function is provided, it will use that function on the Intensities of the reference feature in all samples before normalization and multiply the intensity of each feature with that value after dividing by the Intensity of the reference feature. For example, if

`reference_feature_intensity = mean`, it calculates the mean of the Intensities of the reference features across samples before normalization. It then divides the Intensity of each feature by the Intensity of the reference feature in that sample. Finally, it multiplies each Intensity with the mean of the Intensities of the reference features prior to normalization.

Value

A tibble with intensities normalized across samples.

Examples

```
# Divide by the reference feature and make its Intensity 1000 in each sample
toy_metaboscape %>%
  impute_lod() %>%
 normalize_ref(reference_feature = 2, identifier_column = UID, reference_feature_intensity = 1000)

# Divide by the reference feature and make its Intensity the mean of intensities
# of the reference features before normalization
toy_metaboscape %>%
  impute_lod() %>%
 normalize_ref(reference_feature = 2, identifier_column = UID, reference_feature_intensity = mean)
```

`normalize_sum` *Normalize intensities across samples by dividing by the sample sum*

Description

Normalize across samples by dividing feature intensities by the sum of all intensities in a sample, making the sum 1 in all samples.

Important Note

Intensities of individual features will be very small after this normalization approach. It is therefore advised to multiply all intensities with a fixed number (e.g., 1000) after normalization. See this discussion on OMICSForum.ca and the examples below for further information.

Usage

```
normalize_sum(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with intensities normalized across samples.

Examples

```
# Example 1: Normalization only
toy_metaboscape %>%
  normalize_sum()

# Example 2: Multiply with 1000 after normalization
toy_metaboscape %>%
  normalize_sum() %>%
  dplyr::mutate(Intensity = .data$Intensity * 1000)
```

`plot_pca` *Draws a scores or loadings plot or performs calculations necessary to draw them manually*

Description

Performs PCA and creates a Scores or Loadings plot. Basically a wrapper around `pcaMethods::pca` The plot is drawn with ggplot2 and can therefore be easily manipulated afterwards (e.g., changing the theme or the axis labels). Please note that the function is intended to be easy to use and beginner friendly and therefore offers limited ability to fine-tune certain parameters of the resulting plot. If you wish to draw the plot yourself, you can set `return_tbl = TRUE`. In this case, a tibble is returned instead of a ggplot2 object which you can use to create a plot yourself.

Important Note

`plot_pca()` depends on the `pcaMethods` package from Bioconductor. If `metamorphr` was installed via `install.packages()`, dependencies from Bioconductor were not automatically installed. When `plot_pca()` is called without the `pcaMethods` package installed, you should be asked if you want to install `pak` and `pcaMethods`. If you want to use `plot_pca()` you have to install those. In case you run into trouble with the automatic installation, please install `pcaMethods` manually. See pcaMethods – a Bioconductor package providing PCA methods for incomplete data for instructions on manual installation.

Usage

```
plot_pca(
  data,
  method = "svd",
  what = "scores",
  n_pcs = 2,
  pcs = c(1, 2),
  center = TRUE,
  group_column = NULL,
  name_column = NULL,
  return_tbl = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`method`	A character specifying one of the available methods ("svd", "nipals", "rnipals", "bpca", "ppca", "svdImpute", "robustPca", "nlpca", "llsImpute", "llsImputeAll"). If the default is used ("svd") an SVD PCA will be done, in case `data` does not contain missing values, or a NIPALS PCA if `data` does contain missing values.
`what`	Specifies what should be returned. Either `"scores"` or `"loadings"`.
`n_pcs`	The number of PCs to calculate.
`pcs`	A vector containing 2 integers that specifies the PCs to plot. Only relevant if `return_tbl = FALSE`. The following condition applies: `max(pcs) <= n_pcs`.
`center`	Should `data` be mean centered? See `prep` for details.
`group_column`	Either `NULL` or a column in `data` (e.g., `group_column = Group`). If provided, the dots in the scores plot will be colored according to their group. Only relevant if `what = "scores"`.
`name_column`	Either `NULL` or a column in `data` (e.g., `name_column = Feature`). If provided, feature names are preserved in the resulting tibble. Only relevant if `what = "loadings" & return_tbl = TRUE`.
`return_tbl`	A logical. If `FALSE`, returns a ggplot2 object, if `TRUE` returns a tibble which can be used to draw the plot manually to have more control.
`verbose`	Should outputs from `pca` be printed to the console?

Value

Either a Scores or Loadings Plot in the form of a ggplot2 object or a tibble.

Examples

```
# Draw a Scores Plot
toy_metaboscape %>%
  impute_lod() %>%
  join_metadata(toy_metaboscape_metadata) %>%
  plot_pca(what = "scores", group_column = Group)

# Draw a Loadings Plot
toy_metaboscape %>%
  impute_lod() %>%
  join_metadata(toy_metaboscape_metadata) %>%
  plot_pca(what = "loadings", name_column = Feature)
```

`plot_volcano` *Draws a Volcano Plot or performs calculations necessary to draw one manually*

Description

Performs necessary calculations (i.e., calculate *p*-values and log2-fold changes) and creates a basic Volcano Plot. The plot is drawn with ggplot2 and can therefore be easily manipulated afterwards (e.g., changing the theme or the axis labels). Please note that the function is intended to be easy to use and beginner friendly and therefore offers limited ability to fine-tune certain parameters of the resulting plot. If you wish to draw the plot yourself, you can set `return_tbl = TRUE`. In this case, a tibble is returned instead of a ggplot2 object which you can use to create a plot yourself. A Volcano Plot is used to compare two groups. Therefore grouping information must be provided. See `join_metadata` for more information.

Usage

```
plot_volcano(
  data,
  group_column,
  name_column,
  groups_to_compare,
  batch_column = NULL,
  batch = NULL,
  log2fc_cutoff = 1,
  p_value_cutoff = 0.05,
  colors = list(sig_up = "darkred", sig_down = "darkblue", not_sig_up = "grey",
    not_sig_down = "grey", not_sig = "grey"),
  adjust_p = FALSE,
  log2_before = FALSE,
  return_tbl = FALSE,
  ...
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `group_column = Group`. Uses `args_data_masking`.
`name_column`	Which column contains the feature names? Can for example be `name_column = UID` or `name_column = Feature`. Uses `args_data_masking`.
`groups_to_compare`	Names of the groups which should be compared as a character vector. Those are the group names in the `group_column`. They are usually provided in the form of a metadata tibble and joined via `join_metadata`.

`batch_column`	Which column contains the batch information? Usually `grouping_column = Batch`. Only relevant if `data` contains multiple batches. For example, if `data` contains 2 batches and each batch contains measurements of separate controls, `group_column` and `batch` arguments should be provided. Otherwise controls of both batches will be considered when calculating the *p*-value and log2 fold change. Uses `args_data_masking`.
`batch`	The names of the batch(es) that should be included when calculating *p*-value and log2 fold change.
`log2fc_cutoff`	A numeric. What cutoff should be used for the log2 fold change? Traditionally, this is set to `1` which corresponds to a doubling or halving of intensity or area compared to a control. This is only important for assignment to groups and colors defined in the `colors` argument.
`p_value_cutoff`	A numeric. What cutoff should be used for the *p*-value? Traditionally, this is set to `0.05`. This is only important for assignment to groups and colors defined in the `colors` argument. Note that this is not the -log10 transformed value.
`colors`	A named list for coloring the dots in the Volcano Plot or `NULL` in case the points should not be colored. The list must contain colors for the following groups: `sig_up`, `sig_down`, `not_sig_up`, `not_sig_down` and `not_sig`.
`adjust_p`	Should the *p*-value be adjusted? Can be either `FALSE`, (the default) in case no adjustment should be made or any or the name from `p.adjust.methods` (e.g., `adjust_p = "fdr"`).
`log2_before`	A logical. Should the data be log2 transformed prior to calculating the *p*-values?
`return_tbl`	A logical. If `FALSE`, returns a ggplot2 object, if `TRUE` returns a tibble which can be used to draw the plot manually to have more control.
`...`	Arguments passed on to `t.test`. If none are provided (the default), a Welch Two Sample *t*-test will be performed.

Value

Either a Volcano Plot in the form of a ggplot2 object or a tibble.

Examples

```
# returns a Volcano Plot in the form of a ggplot2 object
toy_metaboscape %>%
  impute_lod() %>%
  join_metadata(toy_metaboscape_metadata) %>%
  plot_volcano(
    group_column = Group,
    name_column = Feature,
    groups_to_compare = c("control", "treatment")
  )

# returns a tibble to draw the plot manually
toy_metaboscape %>%
  impute_lod() %>%
  join_metadata(toy_metaboscape_metadata) %>%
  plot_volcano(
```

```
    group_column = Group,
    name_column = Feature,
    groups_to_compare = c("control", "treatment"),
    return_tbl = TRUE
  )
```

`read_featuretable` *Read a feature table into a tidy tibble*

Description

Basically a wrapper around `readr::read_delim()` but performs some initial tidying operations such as `gather()` rearranging columns. The `label_col` will be renamed to *Feature*.

Usage

```
read_featuretable(file, delim = ",", label_col = 1, metadata_cols = NULL, ...)
```

Arguments

`file`	A path to a file but can also be a connection or literal data.
`delim`	The field separator or delimiter. For example "," in csv files.
`label_col`	The index or name of the column that will be used to label Features. For example an identifier (*e.g.*, KEGG, CAS, HMDB) or a *m/z*-RT pair.
`metadata_cols`	The index/indices or name(s) of column(s) that hold additional feature metadata (*e.g.*, retention times, additional identifiers or *m/z* values).
`...`	Additional arguments passed on to `readr::read_delim()`

Value

A tidy tibble.

References

- H. Wickham, *J. Stat. Soft.* **2014**, *59*, DOI 10.18637/jss.v059.i10.
- H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, H. Yutani, *JOSS* **2019**, *4*, 1686, DOI 10.21105/joss.01686.
- "12 Tidy data | R for Data Science," can be found under `https://r4ds.had.co.nz/tidy-data.html`, **2023**.

Examples

```
# Read a toy dataset in the format produced with Bruker MetaboScape (Version 2021).
featuretable_path <- system.file("extdata", "toy_metaboscape.csv", package = "metamorphr")

# Example 1: Provide indices for metadata_cols
featuretable <- read_featuretable(featuretable_path, metadata_cols = 2:5)

featuretable

# Example 2: Provide a name for label_col and indices for metadata_cols
featuretable <- read_featuretable(
  featuretable_path,
  label_col = "m/z",
  metadata_cols = c(1, 2, 4, 5)
)

featuretable

# Example 3: Provide names for both, label_col and metadata_cols
featuretable <- read_featuretable(
  featuretable_path,
  label_col = "m/z",
  metadata_cols = c("Bucket label", "RT", "Name", "Formula")
)

featuretable
```

`read_mgf` *Read a MGF file into a tidy tibble*

Description

MGF files allow the storage of MS/MS spectra. With this function they can be read into a tidy tibble. Each variable is stored in a column and each ion (observation) is stored in a separate row. MS/MS spectra are stored in a list column named MSn. Please note that MGF files are software-specific so the variables and their names may vary. This function was developed with the GNPS file format exported from mzmine in mind.

Usage

```
read_mgf(file, show_progress = TRUE)
```

Arguments

`file`	The path to the MGF file.
`show_progress`	A `logical` indicating whether the progress of the import should be printed to the console. Only important for large MGF files.

Value

A tidy tibble holding MS/MS spectra.

Examples

```
mgf_path <- system.file("extdata", "toy_mgf.mgf", package = "metamorphr")
read_mgf(mgf_path)
```

`scale_auto` *Scale intensities of features using autoscale*

Description

Scales the intensities of all features using

$$\widetilde{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{s_i}$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling, $\overline{x}_i$ is the mean of intensities of feature i across all samples and s_i is the standard deviation of intensities of feature i across all samples. In other words, it subtracts the mean intensity of a feature across samples from the intensities of that feature in each sample and divides by the standard deviation of that feature. For more information, see the reference section.

Usage

```
scale_auto(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with autoscaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  scale_auto()
```

`scale_center` *Center intensities of features around zero*

Description

Centers the intensities of all features around zero using

$$\widetilde{x}_{ij} = x_{ij} - \overline{x}_i$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling and $\overline{x}_i$ is the mean of intensities of feature i across all samples. In other words, it subtracts the mean intensity of a feature across samples from the intensities of that feature in each sample. For more information, see the reference section.

Usage

```
scale_center(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with intensities scaled around zero.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  scale_center()
```

`scale_level` *Scale intensities of features using level scaling*

Description

Scales the intensities of all features using

$$\widetilde{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{\overline{x}_i}$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling and $\overline{x}_i$ is the mean of intensities of feature i across all samples

In other words, it performs centering (`scale_center`) and divides by the feature mean, thereby focusing on the relative intensity.

Usage

```
scale_level(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with level scaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  impute_lod() %>%
  scale_level()
```

`scale_pareto` *Scale intensities of features using Pareto scaling*

Description

Scales the intensities of all features using

$$\widetilde{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{\sqrt{s_i}}$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling, $\overline{x}_i$ is the mean of intensities of feature i across all samples and $\sqrt{s_i}$ is the square root of the standard deviation of intensities of feature i across all samples. In other words, it subtracts the mean intensity of a feature across samples from the intensities of that feature in each sample and divides by the square root of the standard deviation of that feature. For more information, see the reference section.

Usage

```
scale_pareto(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with autoscaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  scale_pareto()
```

`scale_range` *Scale intensities of features using range scaling*

Description

Scales the intensities of all features using

$$\widetilde{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{x_{i,max} - x_{i,min}}$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling, $\overline{x}_i$ is the mean of intensities of feature i across all samples, $x_{i,max}$ is the maximum intensity of feature i across all samples and $x_{i,min}$ is the minimum intensity of feature i across all samples. In other words, it subtracts the mean intensity of a feature across samples from the intensities of that feature in each sample and divides by the range of that feature. For more information, see the reference section.

Usage

```
scale_range(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with range scaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  scale_range()
```

`scale_vast` *Scale intensities of features using vast scaling*

Description

Scales the intensities of all features using

$$\widetilde{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{s_i} \cdot \frac{\overline{x}_i}{s_i}$$

where $\widetilde{x}_{ij}$ is the intensity of sample j, feature i after scaling, x_{ij} is the intensity of sample j, feature i before scaling, $\overline{x}_i$ is the mean of intensities of feature i across all samples and s_i is the standard deviation of intensities of feature i across all samples. Note that $\frac{\overline{x}_i}{s_i} = \frac{1}{CV}$ where CV is the coefficient of variation across all samples. `scale_vast_grouped` is a variation of this function that uses a group-specific coefficient of variation. In other words, it performs autoscaling (`scale_auto`) and divides by the coefficient of variation, thereby reducing the importance of features with a poor reproducibility.

Usage

```
scale_vast(data)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.

Value

A tibble with vast scaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.
- J. Sun, Y. Xia, *Genes & Diseases* **2024**, *11*, 100979, DOI 10.1016/j.gendis.2023.04.018.

Examples

```
toy_metaboscape %>%
  scale_vast()
```

`scale_vast_grouped` *Scale intensities of features using grouped vast scaling*

Description

A variation of `scale_vast` but uses a group-specific coefficient of variation and therefore requires group information. See `scale_vast` and the References section for more information.

Usage

```
scale_vast_grouped(data, group_column = .data$Group)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`group_column`	Which column should be used for grouping? Usually `grouping_column = Group`. Uses `args_data_masking`.

Value

A tibble with vast scaled intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
  scale_vast_grouped()
```

`summary_featuretable` *General information about a feature table and sample-wise summary*

Description

Information about a feature table. Prints information to the console (number of samples, number of features and if applicable number of groups, replicates and batches) and returns a sample-wise summary as a list.

Usage

```
summary_featuretable(
  data,
  n_samples_max = 5,
  n_features_max = 5,
  n_groups_max = 5,
  n_batches_max = 5
)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n_samples_max`	How many Samples should be printed to the console?
`n_features_max`	How many Features should be printed to the console?
`n_groups_max`	How many groups should be printed to the console?
`n_batches_max`	How many Batches should be printed to the console?

Value

A sample-wise summary as a list.

Examples

```
toy_metaboscape %>%
  join_metadata(toy_metaboscape_metadata) %>%
  summary_featuretable()
```

`toy_metaboscape` *A small toy data set created from a feature table in MetaboScape style*

Description

The raw feature table is also included. This tibble can be reproduced with `metamorphr::read_featuretable(system.file` `"toy_metaboscape.csv", package = "metamorphr"), metadata_cols = 2:5)`.

Usage

```
toy_metaboscape
```

Format

`toy_metaboscape`**:**

A data frame with 110 rows and 8 columns:

UID A unique identifier for each Feature. This column is automatically generated by `metamorphr::read_featuretable(` when the feature table is imported.

Feature A label given to each Feature for easier identification. The column of the original feature table that is used to generate the Feature column is specified with the `label_col` argument of `metamorphr::read_featuretable()`.

Sample Sample name. Column names in the original feature table.

Intensity Measured intensity (or area).

RT Retention time. Feature metadata and therefore not really necessary.

m/z Mass over charge. Feature metadata and therefore not really necessary.

Name Feature name. Feature metadata and therefore not really necessary.

Formula Chemical formula. Feature metadata and therefore not really necessary. ...

Source

This data set contains fictional data!

`toy_metaboscape_metadata`

Sample metadata for the fictional dataset `toy_metaboscape`

Description

Data was generated with `metamorphr::create_metadata_skeleton()` and can be reproduced with `metamorphr::toy_metaboscape %>% create_metadata_skeleton()`.'

Usage

```
toy_metaboscape_metadata
```

Format

`toy_metaboscape_metadata`**:**

A data frame with 11 rows and 5 columns:

Sample The sample name

Group To which group does the samples belong? For example a treatment or a background. Note that additional columns with additional grouping information can be freely added if necessary.

Replicate The replicate.

Batch The batch in which the samples were prepared or measured.

Factor A sample-specific factor, for example dry weight or protein content. ...

Source

This data set contains fictional data!

`toy_mgf` *A small toy data set containing MSn spectra*

Description

Data was generated with `metamorphr::read_mgf()` and can be reproduced with This tibble can be reproduced with `metamorphr::read_mgf(system.file("extdata", "toy_mgf.mgf", package = "metamorphr"))`.

Usage

```
toy_mgf
```

Format

`toy_mgf`**:**

A data frame with 3 rows and 5 columns:

VARIABLEONE A fictional variable.

VARIABLETWO A fictional variable.

VARIABLETHREE A fictional variable.

PEPMASS The precursor ion m/z.

MSn A list column containing MSn spectra. ...

Source

This data set contains fictional data!

`transform_log` *Transforms the intensities by calculating their log*

Description

Log-transforms intensities. The default (base = 10) calculates the log10. This transformation can help reduce heteroscedasticity. See references for more information.

Usage

```
transform_log(data, base = 10)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`base`	Which base should be used for the log-transformation. The default (10) means that log10 values of the intensities are calculated.

Value

A tibble with log-transformed intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  impute_lod() %>%
  transform_log()
```

`transform_power` *Transforms the intensities by calculating their* n*th root*

Description

Calculates the *n*th root of intensities with x^(1/n). The default (n = 2) calculates the square root. This transformation can help reduce heteroscedasticity. See references for more information.

Usage

```
transform_power(data, n = 2)
```

Arguments

`data`	A tidy tibble created by `read_featuretable`.
`n`	The *n*th root to calculate.

Value

A tibble with power-transformed intensities.

References

- R. A. Van Den Berg, H. C. Hoefsloot, J. A. Westerhuis, A. K. Smilde, M. J. Van Der Werf, *BMC Genomics* **2006**, *7*, 142, DOI 10.1186/1471-2164-7-142.

Examples

```
toy_metaboscape %>%
  impute_lod() %>%
  transform_power()
```

Index