# Package 'maaslin3'

October 24, 2025

**Title** ``Refining and extending generalized multivariate linear models for meta-omic association discovery"

**Year** 2025

**Version** 1.1.2

**Depends** R (>= 4.4)

**Description** MaAsLin 3 refines and extends generalized multivariate linear models for meta-omicron association discovery. It finds abundance and prevalence associations between microbiome meta-omics features and complex metadata in population-scale epidemiological studies. The software includes multiple analysis methods (including support for multiple covariates, repeated measures, and ordered predictors), filtering, normalization, and transform options to customize analysis for your specific study.

**License** MIT + file LICENSE

**Imports** dplyr, plyr, pbapply, lmerTest, parallel, lme4, optparse, logging, multcomp, ggplot2, RColorBrewer, patchwork, scales, rlang, tibble, ggnewscale, survival, methods, BiocGenerics, SummarizedExperiment, TreeSummarizedExperiment

**Suggests** knitr, testthat (>= 2.1.0), rmarkdown, markdown, kableExtra

**VignetteBuilder** knitr

**Collate** fit.R utility_scripts.R viz.R maaslin3.R

**URL** http://huttenhower.sph.harvard.edu/maaslin3

**biocViews** Metagenomics, Software, Microbiome, Normalization, MultipleComparison

**BugReports** https://github.com/biobakery/maaslin3/issues

**NeedsCompilation** no

**git_url** https://git.bioconductor.org/packages/maaslin3

**git_branch** devel

**git_last_commit** e7106d5

**git_last_commit_date** 2025-10-12

**Repository** Bioconductor 3.23

**Date/Publication** 2025-10-24

**Author** William Nickols [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8214-9746>),
Jacob Nearing [aut]

**Maintainer** William Nickols <willnickols@g.harvard.edu>

# Contents



---

`maaslin3` *MaAsLin 3: A multivariable statistical framework for finding abundance and prevalence associations between metadata and high-dimensional microbial multi-omics data.*

---

## Description

This wrapper for all MaAsLin 3 steps finds abundance and prevalence associations between microbiome meta-omics features and complex metadata in population-scale epidemiological studies. The software includes multiple analysis methods (including support for multiple covariates, repeated measures, and ordered predictors), filtering, normalization, and transform options to customize analysis for your specific study.

## Usage

```
maaslin3(input_data,
    input_metadata = NULL,
    output,
    formula = NULL,
    fixed_effects = NULL,
    reference = NULL,
    random_effects = NULL,
    group_effects = NULL,
    ordered_effects = NULL,
    strata_effects = NULL,
    feature_specific_covariate = NULL,
    feature_specific_covariate_name = NULL,
    feature_specific_covariate_record = NULL,
    min_abundance = 0,
    min_prevalence = 0,
    max_prevalence = 1.01,
    zero_threshold = 0,
    min_variance = 0,
    max_significance = 0.1,
    normalization = 'TSS',
    transform = 'LOG',
    correction = 'BH',
    standardize = TRUE,
    unscaled_abundance = NULL,
    median_comparison_abundance = TRUE,
    median_comparison_prevalence = FALSE,
    median_comparison_abundance_threshold = 0,
    median_comparison_prevalence_threshold = 0,
    subtract_median = FALSE,
    warn_prevalence = TRUE,
    small_random_effects = FALSE,
    augment = TRUE,
    evaluate_only = NULL,
    plot_summary_plot = TRUE,
    summary_plot_first_n = 25,
    coef_plot_vars = NULL,
    heatmap_vars = NULL,
    plot_associations = TRUE,
    max_pngs = 30,
    cores = 1,
    save_models = FALSE,
    save_plots_rds = FALSE,
    verbosity = 'FINEST',
    summary_plot_balanced = FALSE,
    assay.type = 1)
```

## Arguments

| | |
|---|---|
| input_data | A data frame of feature abundances or read counts, a filepath to a tab-delimited file with abundances, or a SummarizedExperiment or TreeSummarizedExperiment object with the taxa table in 'assays' and metadata in 'colData'. If a data frame or a filepath is supplied, the table should be formatted with features as columns and samples as rows (or the transpose). The column and row names should be the feature names and sample names respectively. |
| input_metadata | A data frame of per-sample metadata or a filepath to a tab-delimited file with metadata. It should be formatted with variables as columns and samples as rows (or the transpose). The column and row names should be the variable names and sample names respectively. |
| output | The output folder to write results. |
| formula | A formula in lme4 format. Random effects, interactions, and functions of the metadata can be included (note that these functions will be applied after standardization if standardize=TRUE). Group, ordered, and strata variables can be specified as: group(grouping_variable), ordered(ordered_variable) and strata(strata_variable). The other variable options below will not be considered if a formula is set. |
| fixed_effects | A vector of variable names to be included as fixed effects. |
| reference | For a variable with more than two levels supplied with fixed_effects, the factor to use as a reference provided as a string of 'variable,reference' semicolon delimited for multiple variables. |
| random_effects | A vector of variable names to be included as random intercepts. |
| group_effects | A factored categorical variable to be included for group testing. An ANOVA-style test will be performed to assess whether any of the variable's levels are significant, and no coefficients or individual p-values will be returned. |
| ordered_effects | A factored categorical variable to be included. Consecutive levels will be tested for significance against each other, and the resulting associations will correspond to effect sizes, standard errors, and significances of each level versus the previous. |
| strata_effects | A vector with one variable name to be included as the strata variable in case-control studies. Strata cannot be combined with random effects. |
| feature_specific_covariate | A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the input_data: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by 'preprocess_mgx_mtx' or 'preprocess_taxa_mtx' first. |
| feature_specific_covariate_name | The name for the feature-specific covariates when fitting the models. This string must be parse-able in a formula (e.g., no spaces). |
| feature_specific_covariate_record | Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots. |

`min_abundance` Features with abundances more than `min_abundance` in more than `min_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation.

`min_prevalence` See `min_abundance`.

`max_prevalence` Features with abundances more than `min_abundance` in fewer than `max_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation.

`zero_threshold` Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives.

`min_variance` Features with abundance variances less than or equal to `min_variance` will be dropped. This is primarily used for dropping features that are entirely zero.

`max_significance`
The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot.

`normalization` The normalization to apply to the features before transformation and analysis. The option `TSS` (total sum scaling) is recommended, but `CLR` (centered log ratio) and `NONE` can also be used.

`transform` The transformation to apply to the features after normalization and before analysis. The option `LOG` (base 2) is recommended, but `PLOG` (pseudo-log) and `NONE` can also be used.

`correction` The correction to obtain FDR-corrected q-values from raw p-values. Any valid options for `p.adjust` can be used.

`standardize` Whether to apply z-scores to continuous metadata variables so they are on the same scale. This is recommended in order to compare coefficients across metadata variables, but note that functions of the metadata specified in the `formula` will apply after standardization.

`unscaled_abundance`
A data frame with a single column of absolute abundances or a filepath to such a tab-delimited file. The row names should match the names of the samples in `input_data` and `input_metadata`. When using spike-ins, the single column should have the same name as one of the features in `input_data`, and the `unscaled_abundance` should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the `unscaled_abundance` should correspond to the total abundance of each sample.

`median_comparison_abundance`
Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data.

`median_comparison_prevalence`
Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity.

median_comparison_abundance_threshold
: Coefficients within median_comparison_abundance_threshold of the median association will automatically be counted as insignificant (p-value set to 1) since they likely represent compositionality-induced associations. This threshold will be divided by the metadata variable's standard deviation if the metadatum is continuous to ensure the threshold applies to the right scale.

median_comparison_prevalence_threshold
: Same as median_comparison_abundance_threshold but applied to the prevalence associations.

subtract_median
: Subtract the median from the coefficients.

warn_prevalence
: Warn when prevalence associations are likely induced by abundance associations. This requires re-fitting the linear models on the TSS log-transformed data.

small_random_effects
: Automatically replace random effects with fixed effects in the logistic prevalence model to handle low numbers of observations per group.

augment
: Add extra lowly-weighted 0s and 1s to avoid linear separability.

evaluate_only
: Whether to evaluate just the abundnace ("abundance") or prevalence ("prevalence") models

plot_summary_plot
: Generate a summary plot of significant associations.

summary_plot_first_n
: Include the top summary_plot_first_n features with significant associations.

coef_plot_vars
: Vector of variable names to be used in the coefficient plot section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form "[variable] [level]".

heatmap_vars
: Vector of variable names to be used in the heatmap section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form "[variable] [level]".

plot_associations
: Whether to generate plots for significant associations.

max_pngs
: The top max_pngs significant associations will be plotted.

cores
: How many cores to use when fitting models. (Using multiple cores will likely be faster only for large datasets or complex models.

save_models
: Whether to return the fit models and save them to an RData file.

save_plots_rds
: Whether to return the fit models and save them to an RData file.

verbosity
: The level of verbosity for the logging package.

summary_plot_balanced
: If set to TRUE the summary plot will show the top N features of each variable included in coef_plot_vars where N is equal to: ceiling(summary_plot_first_n/length(coef_plot_
Will error if coef_plot_vars = NULL

assay.type
: A string or index to select the assay when using a SummarizedExperiment object

## Value

A list containing the following items:

**(1)** `data`: A dataframe of feature abundances with the retained samples for fitting.

**(2)** `normalized_data`: A dataframe of normalized feature abundances.

**(3)** `filtered_data`: A dataframe of feature abundances on the original scale after normalization and filtering.

**(4)** `transformed_data`: A dataframe of feature abundances after filtering, normalization, and transformation.

**(5)** `metadata`: A dataframe of metadata with the retained samples for fitting.

**(6)** `standardized_metadata`: A dataframe of metadata after scaling (if selected).

**(7)** `formula`: Checked or constructed formula(s) specifying the model to be fit.

**(8)** `fit_data_abundance`: The results from the fit abundance models (see `maaslin_fit`).

**(9)** `fit_data_prevalence`: The results from the fit prevalence models (see `maaslin_fit`).

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
fit_out <- maaslin3::maaslin3(input_data = taxa_table,
                              input_metadata = metadata,
                              output = 'output',
                              formula = '~ diagnosis + dysbiosis_state +
                              antibiotics + age + reads',
```

```
                                plot_summary_plot = FALSE,
                                plot_associations = FALSE)

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_check_arguments` *Check parameter arguments to ensure a successful MaAsLin 3 run.*

---

## Description

Check the arguments provided are valid for further MaAsLin 3 use.

## Usage

```
maaslin_check_arguments(feature_specific_covariate = NULL,
                        feature_specific_covariate_name = NULL,
                        feature_specific_covariate_record = NULL,
                        zero_threshold = 0,
                        normalization = 'TSS',
                        transform = 'LOG',
                        correction = 'BH',
                        warn_prevalence = TRUE,
                        evaluate_only = NULL,
                        unscaled_abundance = NULL,
                        median_comparison_abundance = TRUE)
```

## Arguments

`feature_specific_covariate`
: A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the `input_data`: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by ‘preprocess_mgx_mtx‘ or ‘preprocess_taxa_mtx‘ first.

`feature_specific_covariate_name`
: The name for the feature-specific covariates when fitting the models.

`feature_specific_covariate_record`
: Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots.

`zero_threshold`
: Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives.

`normalization` The normalization to apply to the features before transformation and analysis. The option `TSS` (total sum scaling) is recommended, but `CLR` (centered log ratio) and `NONE` can also be used.

`transform` The transformation to apply to the features after normalization and before analysis. The option `LOG` (base 2) is recommended, but `PLOG` (pseudo-log) and `NONE` can also be used.

`correction` The correction to obtain FDR-corrected q-values from raw p-values. Any valid options for `p.adjust` can be used.

`warn_prevalence`
Warn when prevalence associations are likely induced by abundance associations. This requires re-fitting the linear models on the TSS log-transformed data.

`evaluate_only` Whether to evaluate just the abundnace ("abundance") or prevalence ("prevalence") models

`unscaled_abundance`
A data frame with a single column of absolute abundances or a filepath to such a tab-delimited file. The row names should match the names of the samples in `input_data` and `input_metadata`. When using spike-ins, the single column should have the same name as one of the features in `input_data`, and the `unscaled_abundance` should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the `unscaled_abundance` should correspond to the total abundance of each sample.

`median_comparison_abundance`
Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data.

## Value

No value is returned, but incompatibile arguments will produce an error.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
```

```
    "maaslin3")
    metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

    metadata$diagnosis <-
        factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
    metadata$dysbiosis_state <-
        factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
        'dysbiosis_CD'))
    metadata$antibiotics <-
        factor(metadata$antibiotics, levels = c('No', 'Yes'))

    # Prepare parameter lists
    maaslin3::maaslin_check_arguments(zero_threshold = 0,
                                      normalization = 'TSS',
                                      transform = 'LOG',
                                      correction = 'BH',
                                      median_comparison_abundance = TRUE)

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_check_formula` *Check a MaAsLin 3 formula to ensure a proper MaAsLin 3 run.*

---

## Description

Ensure that the formula provided is valid. Only one of `maaslin_compute_formula` or `maaslin_check_formula` should be used.

## Usage

```
maaslin_check_formula(data,
                      metadata,
                      input_formula = NULL,
                      feature_specific_covariate_name = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `metadata` | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| `input_formula` | A formula in `lme4` format. Random effects, interactions, and functions of the metadata can be included (note that these functions will be applied after standardization if `standardize=TRUE`). Group, ordered, and strata variables can be specified as: `group(grouping_variable)`, `ordered(ordered_variable)` and |

strata(strata_variable). The other variable options below will not be considered if a formula is set.

feature_specific_covariate_name
: The name for the feature-specific covariates when fitting the models.

## Value

A list containing the following named items:

**(1)** formula: The constructed formula.

**(2)** random_effects_formula: A formula for the random effects.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)
    read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
```

```
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_compute_formula` *Compute a formula for a MaAsLin 3 run based on the specified effects.*

---

## Description

Compute a formula using variables provided through `fixed_effects`, `random_effects`, `group_effects`, `ordered_effects`, and `strata_effects`. Only one of `maaslin_compute_formula` or `maaslin_check_formula` should be used.

## Usage

```
maaslin_compute_formula(data,
                        metadata,
                        fixed_effects = NULL,
                        random_effects = NULL,
                        group_effects = NULL,
                        ordered_effects = NULL,
                        strata_effects = NULL,
                        feature_specific_covariate_name = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `metadata` | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| `fixed_effects` | A vector of variable names to be included as fixed effects. |
| `random_effects` | A vector of variable names to be included as random intercepts. |

`group_effects` A factored categorical variable to be included for group testing. An ANOVA-style test will be performed to assess whether any of the variable's levels are significant, and no coefficients or individual p-values will be returned.

`ordered_effects`
A factored categorical variable to be included. Consecutive levels will be tested for significance against each other, and the resulting associations will correspond to effect sizes, standard errors, and significances of each level versus the previous.

`strata_effects` A vector with one variable name to be included as the strata variable in case-control studies. Strata cannot be combined with random effects.

`feature_specific_covariate_name`
The name for the feature-specific covariates when fitting the models.

## Value

A list containing the following named items:

**(1)** `formula`: The constructed formula.

**(2)** `random_effects_formula`: A formula for the random effects.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
```

```
    input_metadata = metadata,
    output = 'output',
    fixed_effects = c('diagnosis', 'dysbiosis_state', 'antibiotics',
                      'age', 'reads'),
    random_effects = c('participant_id'),
    plot_summary_plot = FALSE,
    plot_associations = FALSE)
read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_compute_formula(
    data,
    metadata,
    fixed_effects = c('diagnosis', 'dysbiosis_state', 'antibiotics',
                      'age', 'reads'),
    random_effects = c('participant_id'))

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_contrast_test` *Perform a contrast from a fit MaAsLin 3 model.*

---

## Description

Perform a contrast test (`lmerTest::contest` for mixed effects linear; `multcomp::glht` for all others) using a named contrast matrix and right hand side. One contrast test is applied per row of the matrix.

## Usage

```
maaslin_contrast_test(maaslin3_fit,
                      contrast_mat,
                      rhs = NULL,
                      max_significance = 0.1,
                      correction = 'BH',
                      median_comparison_abundance = TRUE,
                      median_comparison_prevalence = FALSE,
                      subtract_median = FALSE,
                      small_random_effects = FALSE,
                      evaluate_only = NULL)
```

## Arguments

| | |
|---|---|
| maaslin3_fit | The output of maaslin_fit with save_models = TRUE. |
| contrast_mat | A matrix with one row per contrast test to run. The columns will be matched to the coefficients of the model by name. Contrast vector coefficients need not be specified if they would be zero. If row names are provided, they will be used to label the test in the results. |
| rhs | The right hand size of the contrast test. The length should be the same as the number of rows in the contrast_mat. This will default to 0 or the median comparison if median_comparison=TRUE. |
| max_significance | The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot. |
| correction | The correction to obtain FDR-corrected q-values from raw p-values. Any valid options for p.adjust can be used. |
| median_comparison_abundance | Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data. |
| median_comparison_prevalence | Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity. |
| subtract_median | Subtract the median from the coefficients. |
| small_random_effects | Automatically replace random effects with fixed effects in the logistic prevalence model to handle low numbers of observations per group. |
| evaluate_only | Whether to evaluate just the abundnace ("abundance") or prevalence ("prevalence") models |

## Value

A dataframe with the following columns:

**(1)** feature: The feature involved in the association.

**(2)** test: The contrast test name.

**(3)** coef: The coefficient of the association: the slope coefficient in the abundance model and the change in log odds in the prevalence model.

**(4)** null_hypothesis: The value of the null hypothesis against which the coefficients are tested (zero or the per-metadatum median).

**(5)** stderr: The standard error of the coefficient.

**(6)** pval_individual: The (uncorrected) p-value of the association.

**(7)** qval_individual: The FDR corrected q-value of the association. FDR correction is performed over all associations in the abundance and prevalence modeling without errors together.

**(8)** pval_joint: The p-value of the overall association (combining abundance and prevalence) by taking the minimum of the abundance and logistic p-values and applying the Beta(1,2) CDF. These will be the same in the abundance and prevalence results for an association.

**(9)** qval_joint: The FDR corrected q-value of the association. FDR correction is performed over all joint p-values without errors.

**(10)** error: Any error produced by the model during fitting. NA otherwise.

**(11)** model: linear for the abundance models and logistic for the prevalence models.

**(12)** N: The number of data points for the association's feature.

**(13)** N_not_zero: The number of non-zero data points for the association's feature.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
fit_out <- maaslin3::maaslin3(input_data = taxa_table,
                              input_metadata = metadata,
                              output = 'output',
                              formula = '~ diagnosis + dysbiosis_state +
                              antibiotics + age + reads',
                              plot_summary_plot = FALSE,
                              plot_associations = FALSE)

contrast_mat <- matrix(c(1, 1, 0, 0, 0, 0, 1, 1),
    ncol = 4, nrow = 2, byrow = TRUE)

colnames(contrast_mat) <- c("diagnosisUC",
```

```
                                "diagnosisCD",
                                "dysbiosis_statedysbiosis_UC",
                                "dysbiosis_statedysbiosis_CD")

    rownames(contrast_mat) <- c("diagnosis_test", "dysbiosis_test")

    maaslin_contrast_test(maaslin3_fit = fit_out,
                          contrast_mat = contrast_mat)

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_filter` *Filter abundance data before MaAsLin 3 model fitting.*

---

## Description

Set abundances below `zero_threshold` to zero, remove features without abundances more than `min_abundance` in `min_prevalence` of the samples, remove features with abundances more than `min_abundance` in more than `max_prevalence` of the samples, and remove features with variances less than or equal to `min_variance`.

## Usage

```
maaslin_filter(normalized_data,
            output,
            min_abundance = 0,
            min_prevalence = 0,
            max_prevalence = 1.01,
            zero_threshold = 0,
            min_variance = 0)
```

## Arguments

| | |
|---|---|
| `normalized_data` | A data frame of normalized feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `output` | The output folder to write results. |
| `min_abundance` | Features with abundances more than `min_abundance` in more than `min_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation. |
| `min_prevalence` | See `min_abundance`. |
| `max_prevalence` | Features with abundances more than `min_abundance` in fewer than `max_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation. |

`zero_threshold` Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives.

`min_variance` Features with abundance variances less than or equal to `min_variance` will be dropped. This is primarily used for dropping features that are entirely zero.

## Value

A dataframe of filtered features (features are columns; samples are rows).

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
```

```
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                                output = 'output')

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_fit` *Fit MaAsLin 3 models.*

---

## Description

Fit the abundance data with abundance and prevalence models to discover feature-metadata associations.

## Usage

```
maaslin_fit(filtered_data,
            transformed_data,
            metadata,
            formula,
            random_effects_formula,
            feature_specific_covariate = NULL,
            feature_specific_covariate_name = NULL,
            feature_specific_covariate_record = NULL,
            zero_threshold = 0,
            max_significance = 0.1,
            correction = 'BH',
            median_comparison_abundance = TRUE,
            median_comparison_prevalence = FALSE,
            median_comparison_abundance_threshold = 0,
            median_comparison_prevalence_threshold = 0,
            subtract_median = FALSE,
            warn_prevalence = TRUE,
            small_random_effects = FALSE,
            augment = TRUE,
```

```
            evaluate_only = NULL,
            cores = 1,
            save_models = FALSE,
            data = NULL,
            min_abundance = 0,
            min_prevalence = 0,
            max_prevalence = 1.01,
            min_variance = 0)
```

## Arguments

| | |
|---|---|
| `filtered_data` | A data frame of filtered feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `transformed_data` | A data frame of transformed feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `metadata` | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| `formula` | A formula in `lme4` format as from `maaslin_check_formula`. |
| `random_effects_formula` | A formula in `lme4` format as from `maaslin_check_formula`. |
| `feature_specific_covariate` | A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the `input_data`: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by 'preprocess_mgx_mtx' or 'preprocess_taxa_mtx' first. |
| `feature_specific_covariate_name` | The name for the feature-specific covariates when fitting the models. |
| `feature_specific_covariate_record` | Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots. |
| `zero_threshold` | Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives. |
| `max_significance` | The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot. |
| `correction` | The correction to obtain FDR-corrected q-values from raw p-values. Any valid options for `p.adjust` can be used. |
| `median_comparison_abundance` | Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data. |

`median_comparison_prevalence`
: Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity.

`median_comparison_abundance_threshold`
: Coefficients within `median_comparison_abundance_threshold` of the median association will automatically be counted as insignificant (p-value set to 1) since they likely represent compositionality-induced associations. This threshold will be divided by the metadata variable's standard deviation if the metadatum is continuous to ensure the threshold applies to the right scale.

`median_comparison_prevalence_threshold`
: Same as `median_comparison_abundance_threshold` but applied to the prevalence associations.

`subtract_median`
: Subtract the median from the coefficients.

`warn_prevalence`
: Warn when prevalence associations are likely induced by abundance associations. This requires re-fitting the linear models on the TSS log-transformed data.

`small_random_effects`
: Automatically replace random effects with fixed effects in the logistic prevalence model to handle low numbers of observations per group.

`augment` Add extra lowly-weighted 0s and 1s to avoid linear separability.

`evaluate_only` Whether to evaluate just the abundnace ("abundance") or prevalence ("prevalence") models

`cores` How many cores to use when fitting models. (Using multiple cores will likely be faster only for large datasets or complex models.

`save_models` Whether to return the fit models and save them to an RData file.

`data` The original data (only necessary if warn_prevalence is TRUE).

`min_abundance` The original min_abundance parameter (only necessary if warn_prevalence is TRUE).

`min_prevalence` The original min_prevalence parameter (only necessary if warn_prevalence is TRUE).

`max_prevalence` The original max_prevalence parameter (only necessary if warn_prevalence is TRUE).

`min_variance` The original min_variance parameter (only necessary if min_variance is TRUE).

## Value

A list containing the following named items:

**(1)** `fit_data_abundance`: The results from the fit abundance models.

**(2)** `fit_data_prevalence`: The results from the fit prevalence models.

The `fit_data_abundance` and `fit_data_prevalence` items have the same structure. They are both lists with the following named items:

**(1)** `results`: A results table with the modeled associations (see below).

**(2)** `residuals`: A features (rows) by samples (columns) dataframe of residuals from the models.

**(3)** `fitted`: A features (rows) by samples (columns) dataframe of fitted values from the models.

**(4)** `ranef`: A features (rows) by random effect (columns) dataframe of random effects from the models. If multiple random effects are specified, this is a dataframe of dataframes.

**(5)** `fits`: If `save_models=TRUE`, this is a list of the fit models.

The `results` tables contain the following columns for each association (row):

**(1)** `feature`: The feature involved in the association.

**(2)** `metadata`: The metadata variable involved in the association.

**(3)** `value`: The value of the metadata variable: the metadata variable itself if continuous or the level if categorical.

**(4)** `name`: The name of the model component involved in the association: the metadata variable itself if continuous or a concatenated version of the metadata variable and level if categorical.

**(5)** `coef`: The coefficient of the association: the slope coefficient in the abundance model and the change in log odds in the prevalence model.

**(6)** `null_hypothesis`: The value of the null hypothesis against which the coefficients are tested (zero or the per-metadatum median).

**(7)** `stderr`: The standard error of the coefficient.

**(8)** `pval_individual`: The (uncorrected) p-value of the association.

**(9)** `qval_individual`: The FDR corrected q-value of the association. FDR correction is performed over all associations in the abundance and prevalence modeling without errors together.

**(10)** `pval_joint`: The p-value of the overall association (combining abundance and prevalence) by taking the minimum of the abundance and logistic p-values and applying the Beta(1,2) CDF. These will be the same in the abundance and prevalence results for an association.

**(11)** `qval_joint`: The FDR corrected q-value of the association. FDR correction is performed over all joint p-values without errors.

**(12)** `error`: Any error produced by the model during fitting. NA otherwise.

**(13)** `model`: `linear` for the abundance models and `logistic` for the prevalence models.

**(14)** `N`: The number of data points for the association's feature.

**(15)** `N_not_zero`: The number of non-zero data points for the association's feature.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

formula <- formulas$formula
random_effects_formula <- formulas$random_effects_formula

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
```

```
                                output = 'output')

transformed_data = maaslin3::maaslin_transform(filtered_data,
                                output = 'output')

standardized_metadata = maaslin3::maaslin_process_metadata(
    metadata,
    formula = formula)

maaslin_results = maaslin3::maaslin_fit(
    filtered_data,
    transformed_data,
    standardized_metadata,
    formula,
    random_effects_formula,
    warn_prevalence = FALSE)

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_log_arguments` *Log MaAsLin 3 parameters.*

---

## Description

Check that the parameters provided are valid for further MaAsLin 3 use and open a logger to log the parameters.

## Usage

```
maaslin_log_arguments(input_data,
                      input_metadata,
                      output,
                      formula = NULL,
                      fixed_effects = NULL,
                      reference = NULL,
                      random_effects = NULL,
                      group_effects = NULL,
                      ordered_effects = NULL,
                      strata_effects = NULL,
                      feature_specific_covariate = NULL,
                      feature_specific_covariate_name = NULL,
                      feature_specific_covariate_record = NULL,
                      min_abundance = 0,
                      min_prevalence = 0,
                      max_prevalence = 1.01,
                      zero_threshold = 0,
                      min_variance = 0,
```

```
max_significance = 0.1,
normalization = 'TSS',
transform = 'LOG',
correction = 'BH',
standardize = TRUE,
unscaled_abundance = NULL,
median_comparison_abundance = TRUE,
median_comparison_prevalence = FALSE,
median_comparison_abundance_threshold = 0,
median_comparison_prevalence_threshold = 0,
subtract_median = FALSE,
warn_prevalence = TRUE,
small_random_effects = FALSE,
augment = TRUE,
evaluate_only = NULL,
plot_summary_plot = TRUE,
summary_plot_first_n = 25,
coef_plot_vars = NULL,
heatmap_vars = NULL,
plot_associations = TRUE,
max_pngs = 30,
cores = 1,
save_models = FALSE,
save_plots_rds = FALSE,
verbosity = 'FINEST',
summary_plot_balanced = FALSE)
```

## Arguments

| | |
|---|---|
| input_data | A data frame of feature abundances or read counts or a filepath to a tab-delimited file with abundances. It should be formatted with features as columns and samples as rows (or the transpose). The column and row names should be the feature names and sample names respectively. |
| input_metadata | A data frame of per-sample metadata or a filepath to a tab-delimited file with metadata. It should be formatted with variables as columns and samples as rows (or the transpose). The column and row names should be the variable names and sample names respectively. |
| output | The output folder to write results. |
| formula | A formula in lme4 format. Random effects, interactions, and functions of the metadata can be included (note that these functions will be applied after standardization if standardize=TRUE). Group, ordered, and strata variables can be specified as: group(grouping_variable), ordered(ordered_variable) and strata(strata_variable). The other variable options below will not be considered if a formula is set. |
| fixed_effects | A vector of variable names to be included as fixed effects. |
| reference | For a variable with more than two levels supplied with fixed_effects, the factor to use as a reference provided as a string of 'variable,reference' semicolon delimited for multiple variables. |

`random_effects` A vector of variable names to be included as random intercepts.

`group_effects` A factored categorical variable to be included for group testing. An ANOVA-style test will be performed to assess whether any of the variable's levels are significant, and no coefficients or individual p-values will be returned.

`ordered_effects`
A factored categorical variable to be included. Consecutive levels will be tested for significance against each other, and the resulting associations will correspond to effect sizes, standard errors, and significances of each level versus the previous.

`strata_effects` A vector with one variable name to be included as the strata variable in case-control studies. Strata cannot be combined with random effects.

`feature_specific_covariate`
A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the `input_data`: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by 'preprocess_mgx_mtx' or 'preprocess_taxa_mtx' first.

`feature_specific_covariate_name`
The name for the feature-specific covariates when fitting the models.

`feature_specific_covariate_record`
Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots.

`min_abundance` Features with abundances more than `min_abundance` in more than `min_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation.

`min_prevalence` See `min_abundance`.

`max_prevalence` Features with abundances more than `min_abundance` in fewer than `max_prevalence` of the samples will be included for analysis. The threshold is applied after normalization and before transformation.

`zero_threshold` Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives.

`min_variance` Features with abundance variances less than or equal to `min_variance` will be dropped. This is primarily used for dropping features that are entirely zero.

`max_significance`
The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot.

`normalization` The normalization to apply to the features before transformation and analysis. The option `TSS` (total sum scaling) is recommended, but `CLR` (centered log ratio) and `NONE` can also be used.

`transform` The transformation to apply to the features after normalization and before analysis. The option `LOG` (base 2) is recommended, but `PLOG` (pseudo-log) and `NONE` can also be used.

`correction` The correction to obtain FDR-corrected q-values from raw p-values. Any valid options for `p.adjust` can be used.

`standardize` Whether to apply z-scores to continuous metadata variables so they are on the same scale. This is recommended in order to compare coefficients across metadata variables, but note that functions of the metadata specified in the `formula` will apply after standardization.

`unscaled_abundance`
A data frame with a single column of absolute abundances or a filepath to such a tab-delimited file. The row names should match the names of the samples in `input_data` and `input_metadata`. When using spike-ins, the single column should have the same name as one of the features in `input_data`, and the `unscaled_abundance` should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the `unscaled_abundance` should correspond to the total abundance of each sample.

`median_comparison_abundance`
Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data.

`median_comparison_prevalence`
Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity.

`median_comparison_abundance_threshold`
Coefficients within `median_comparison_abundance_threshold` of the median association will automatically be counted as insignificant (p-value set to 1) since they likely represent compositionality-induced associations. This threshold will be divided by the metadata variable's standard deviation if the metadatum is continuous to ensure the threshold applies to the right scale.

`median_comparison_prevalence_threshold`
Same as `median_comparison_abundance_threshold` but applied to the prevalence associations.

`subtract_median`
Subtract the median from the coefficients.

`warn_prevalence`
Warn when prevalence associations are likely induced by abundance associations. This requires re-fitting the linear models on the TSS log-transformed data.

`small_random_effects`
Automatically replace random effects with fixed effects in the logistic prevalence model to handle low numbers of observations per group.

`augment` Add extra lowly-weighted 0s and 1s to avoid linear separability.

`evaluate_only` Whether to evaluate just the abundnace ("abundance") or prevalence ("prevalence") models

`plot_summary_plot`
Generate a summary plot of significant associations.

summary_plot_first_n
: Include the top summary_plot_first_n features with significant associations.

coef_plot_vars
: Vector of variable names to be used in the coefficient plot section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form "[variable] [level]".

heatmap_vars
: Vector of variable names to be used in the heatmap section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form "[variable] [level]".

plot_associations
: Whether to generate plots for significant associations.

max_pngs
: The top max_pngs significant associations will be plotted.

cores
: How many cores to use when fitting models. (Using multiple cores will likely be faster only for large datasets or complex models.

save_models
: Whether to return the fit models and save them to an RData file.

save_plots_rds
: Whether to return the plots to an RDS file.

verbosity
: The level of verbosity for the logging package.

summary_plot_balanced
: If set to TRUE the summary plot will show the top N features of each variable included in coef_plot_vars where N is equal to: ceiling(summary_plot_first_n/length(coef_plot_
Will error if coef_plot_vars = NULL

## Value

No value is returned, but a logger is opened with the parameters logged.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
```

```
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

# Prepare parameter lists
maaslin3::maaslin_log_arguments(input_data = taxa_table,
                input_metadata = metadata,
                output = 'output',
                formula = '~ diagnosis + dysbiosis_state +
                    antibiotics + age + reads',
                plot_summary_plot = FALSE,
                plot_associations = FALSE)
unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_normalize` *Normalize abundance data for MaAsLin 3 model fitting.*

---

## Description

Normalize the abundance data according to the `normalization` parameter. If `unscaled_abundance` is specified, compute the absolute abundances.

## Usage

```
maaslin_normalize(data,
                output,
                zero_threshold = 0,
                normalization = 'TSS',
                unscaled_abundance = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `output` | The output folder to write results. |
| `zero_threshold` | Abundances less than or equal to `zero_threshold` will be treated as zeros. This is primarily to be used when the abundance table has likely low-abundance false positives. |
| `normalization` | The normalization to apply to the features before transformation and analysis. The option `TSS` (total sum scaling) is recommended, but `CLR` (centered log ratio) and `NONE` can also be used. |
| `unscaled_abundance` | A data frame with a single column of absolute abundances. The row names should match the names of the samples in `input_data` and `input_metadata`. When using spike-ins, the single column should have the same name as one of |

the features in input_data, and the unscaled_abundance should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the unscaled_abundance should correspond to the total abundance of each sample.

## Value

A dataframe of normalized features (features are columns; samples are rows).

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)
```

```
data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_plot_results` *Plot the results from a MaAsLin 3 run.*

---

## Description

Two types of plots are generated. First, the summary plot contains sorted per-feature coefficients plotted with their standard errors for key variables and a heatmap summarizing the remaining variables. Second, for significant features, association plots (scatterplots, boxplots, or tables depending on the association) are generated to visualize and verify the model fits. The data are shown with their transformed values in the association plots since this is the scale on which the models are fit.

## Usage

```
maaslin_plot_results(output,
                     transformed_data,
                     unstandardized_metadata,
                     fit_data_abundance,
                     fit_data_prevalence,
                     normalization,
                     transform,
                     feature_specific_covariate = NULL,
                     feature_specific_covariate_name = NULL,
                     feature_specific_covariate_record = NULL,
                     median_comparison_abundance = TRUE,
                     median_comparison_prevalence = FALSE,
                     max_significance = 0.1,
                     plot_summary_plot = TRUE,
                     summary_plot_first_n = 25,
                     coef_plot_vars = NULL,
                     heatmap_vars = NULL,
                     plot_associations = TRUE,
                     max_pngs = 30,
```

```
                    balanced = FALSE,
                    save_plots_rds = FALSE)
```

## Arguments

| | |
|---|---|
| output | The output folder to write results. |
| transformed_data | A data frame of transformed feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| unstandardized_metadata | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| fit_data_abundance | The abundance outputs of maaslin_fit. |
| fit_data_prevalence | The prevalence outputs of maaslin_fit. |
| normalization | The normalization to apply to the features before transformation and analysis. The option TSS (total sum scaling) is recommended, but CLR (centered log ratio) and NONE can also be used. |
| transform | The transformation to apply to the features after normalization and before analysis. The option LOG (base 2) is recommended, but PLOG (pseudo-log) and NONE can also be used. |
| feature_specific_covariate | A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the input_data: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by ‘preprocess_mgx_mtx‘ or ‘preprocess_taxa_mtx‘ first. |
| feature_specific_covariate_name | The name for the feature-specific covariates when fitting the models. |
| feature_specific_covariate_record | Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots. |
| median_comparison_abundance | Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data. |
| median_comparison_prevalence | Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity. |

`max_significance`
: The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot.

`plot_summary_plot`
: Generate a summary plot of significant associations.

`summary_plot_first_n`
: Include the top `summary_plot_first_n` features with significant associations.

`coef_plot_vars`
: Vector of variable names to be used in the coefficient plot section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form `"[variable] [level]"`.

`heatmap_vars`
: Vector of variable names to be used in the heatmap section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form `"[variable] [level]"`.

`plot_associations`
: Whether to generate plots for significant associations.

`max_pngs`
: The top `max_pngs` significant associations will be plotted.

`balanced`
: If set to TRUE the summary plot will show the top N features of each variable included in `coef_plot_vars` where N is equal to: `ceiling(summary_plot_first_n/length(coef_plot_`
Will error if `coef_plot_vars = NULL`

`save_plots_rds`
: Whether to return the plots to an RDS file.

## Value

Results will be written to the `figures` folder within the folder `output`. The list of individual association plots is returned if `plot_associations=TRUE`. In the heatmap of the summary plot, one star corresponds to the user-set `max_significance` and two stars corresponds to the user-set `max_signifiance` divided by 10.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
```

```
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

formula <- formulas$formula
random_effects_formula <- formulas$random_effects_formula

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                                output = 'output')

transformed_data = maaslin3::maaslin_transform(filtered_data,
                                output = 'output')

standardized_metadata = maaslin3::maaslin_process_metadata(
    metadata,
    formula = formula)

maaslin_results = maaslin3::maaslin_fit(
    filtered_data,
    transformed_data,
    standardized_metadata,
    formula,
    random_effects_formula,
```

```
    warn_prevalence = FALSE)

maaslin3::maaslin_write_results(
    output = 'output',
    maaslin_results$fit_data_abundance,
    maaslin_results$fit_data_prevalence,
    random_effects_formula)

maaslin3::maaslin_plot_results(
    output = 'output',
    transformed_data,
    metadata,
    maaslin_results$fit_data_abundance,
    maaslin_results$fit_data_prevalence,
    normalization = "TSS",
    transform = "LOG")

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_plot_results_from_output` *Plot the results from a MaAsLin 3 run.*

---

## Description

Two types of plots are generated. First, the summary plot contains sorted per-feature coefficients plotted with their standard errors for key variables and a heatmap summarizing the remaining variables. Second, for significant features, association plots (scatterplots, boxplots, or tables depending on the association) are generated to visualize and verify the model fits. The data are shown with their transformed values in the association plots since this is the scale on which the models are fit. In comparison to `maaslin_plot_results` that needs the entire `maaslin_fit` list, only the parameter list and an outputs directory containing a completed run are needed for `maaslin_plot_results_from_output`.

## Usage

```
maaslin_plot_results_from_output(output,
                                 metadata,
                                 normalization,
                                 transform,
                                 feature_specific_covariate = NULL,
                                 feature_specific_covariate_name = NULL,
                                 feature_specific_covariate_record = NULL,
                                 median_comparison_abundance = TRUE,
                                 median_comparison_prevalence = FALSE,
                                 max_significance = 0.1,
                                 plot_summary_plot = TRUE,
                                 summary_plot_first_n = 25,
```

```
                                coef_plot_vars = NULL,
                                heatmap_vars = NULL,
                                plot_associations = TRUE,
                                max_pngs = 30,
                                balanced = FALSE,
                                save_plots_rds = FALSE)
```

## Arguments

| | |
|---|---|
| output | The output folder to write results. |
| metadata | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| normalization | The normalization to apply to the features before transformation and analysis. The option TSS (total sum scaling) is recommended, but CLR (centered log ratio) and NONE can also be used. |
| transform | The transformation to apply to the features after normalization and before analysis. The option LOG (base 2) is recommended, but PLOG (pseudo-log) and NONE can also be used. |
| feature_specific_covariate | A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the input_data: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by 'preprocess_mgx_mtx' or 'preprocess_taxa_mtx' first. |
| feature_specific_covariate_name | The name for the feature-specific covariates when fitting the models. |
| feature_specific_covariate_record | Whether to keep the feature-specific covariates in the outputs when calculating p-values, writing results, and displaying plots. |
| median_comparison_abundance | Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is recommended for relative abundance data but should not be used for absolute abundance data. |
| median_comparison_prevalence | Test prevalence coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. This is only recommended if the analyst is interested in how feature prevalence associations compare to each other or if there is likely strong compositionality-induced sparsity. |
| max_significance | The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot. |
| plot_summary_plot | Generate a summary plot of significant associations. |
| summary_plot_first_n | Include the top summary_plot_first_n features with significant associations. |

`coef_plot_vars` Vector of variable names to be used in the coefficient plot section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form `"[variable] [level]"`.

`heatmap_vars` Vector of variable names to be used in the heatmap section of the summary plot. Continuous variables should match the metadata column name, and categorical variables should be of the form `"[variable] [level]"`.

`plot_associations` Whether to generate plots for significant associations.

`max_pngs` The top `max_pngs` significant associations will be plotted.

`balanced` If set to TRUE the summary plot will show the top N features of each variable included in `coef_plot_vars` where N is equal to: `ceiling(summary_plot_first_n/length(coef_plot_`
Will error if `coef_plot_vars = NULL`

`save_plots_rds` Whether to return the plots to an RDS file.

## Value

Results will be written to the `figures` folder within the folder `output`. The list of individual association plots is returned if `plot_associations=TRUE`. In the heatmap of the summary plot, one star corresponds to the user-set `max_significance` and two stars corresponds to the user-set `max_signifiance` divided by 10.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
```

```
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

formula <- formulas$formula
random_effects_formula <- formulas$random_effects_formula

normalized_data = maaslin3::maaslin_normalize(data,
                            output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                            output = 'output')

transformed_data = maaslin3::maaslin_transform(filtered_data,
                            output = 'output')

standardized_metadata = maaslin3::maaslin_process_metadata(
    metadata,
    formula = formula)

maaslin_results = maaslin3::maaslin_fit(
    filtered_data,
    transformed_data,
    standardized_metadata,
    formula,
    random_effects_formula,
    warn_prevalence = FALSE)

maaslin3::maaslin_write_results(
    output = 'output',
    maaslin_results$fit_data_abundance,
    maaslin_results$fit_data_prevalence,
    random_effects_formula)
```

```
maaslin3::maaslin_plot_results_from_output(
    output = 'output',
    metadata,
    normalization = "TSS",
    transform = "LOG")

unlink('output', recursive=TRUE)
logging::logReset()
```

---

maaslin_process_metadata

*Process metadata before MaAsLin 3 model fitting.*

---

## Description

Check that references are set properly if the metadata variables are categorical and provided through `fixed_effects`. Standardize the continuous metadata variables as a z-score (subtract the mean, divide by the standard deviation) if `standardize` is set.

## Usage

```
maaslin_process_metadata(metadata,
                         formula = NULL,
                         fixed_effects = NULL,
                         reference = NULL,
                         feature_specific_covariate_name = NULL,
                         standardize = TRUE)
```

## Arguments

| | |
|---|---|
| `metadata` | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows. The column and row names should be the variable names and sample names respectively. |
| `formula` | A formula in `lme4` format. Random effects, interactions, and functions of the metadata can be included (note that these functions will be applied after standardization if `standardize=TRUE`). Group, ordered, and strata variables can be specified as: `group(grouping_variable)`, `ordered(ordered_variable)` and `strata(strata_variable)`. The other variable options below will not be considered if a formula is set. |
| `fixed_effects` | A vector of variable names to be included as fixed effects. |
| `reference` | For a variable with more than two levels supplied with `fixed_effects`, the factor to use as a reference provided as a string of 'variable,reference' semicolon delimited for multiple variables. |
| `feature_specific_covariate_name` | The name for the feature-specific covariates when fitting the models. |

`standardize` Whether to apply z-scores to continuous metadata variables so they are on the same scale. This is recommended in order to compare coefficients across metadata variables, but note that functions of the metadata specified in the `formula` will apply after standardization.

## Value

The processed metadata.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)
```

```
data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

formula <- formulas$formula
random_effects_formula <- formulas$random_effects_formula

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                                output = 'output')

standardized_metadata = maaslin3::maaslin_process_metadata(
    metadata,
    formula = formula)

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_read_data` *Read in the abundance data and metadata.*

---

## Description

Read in the abundance data and metadata from files if necessary.

## Usage

```
maaslin_read_data(input_data,
                  input_metadata,
                  feature_specific_covariate = NULL,
                  unscaled_abundance = NULL)
```

## Arguments

| | |
|---|---|
| `input_data` | A data frame of feature abundances or read counts or a filepath to a tab-delimited file with abundances. It should be formatted with features as columns and samples as rows (or the transpose). The column and row names should be the feature names and sample names respectively. |
| `input_metadata` | A data frame of per-sample metadata or a filepath to a tab-delimited file with metadata. It should be formatted with variables as columns and samples as rows (or the transpose). The column and row names should be the variable names and sample names respectively. |

`feature_specific_covariate`
: A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the `input_data`: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by 'preprocess_mgx_mtx' or 'preprocess_taxa_mtx' first.

`unscaled_abundance`
: A data frame with a single column of absolute abundances or a filepath to such a tab-delimited file. The row names should match the names of the samples in `input_data` and `input_metadata`. When using spike-ins, the single column should have the same name as one of the features in `input_data`, and the `unscaled_abundance` should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the `unscaled_abundance` should correspond to the total abundance of each sample.

## Value

A list containing the following items:

**(1)** `data`: A data frame of feature abundances.

**(2)** `metadata`: A data frame of metadata.

**(3)** `feature_specific_covariate`: A data frame of feature specific covariates.

**(4)** `unscaled_abundance`: A data frame of unscaled abundances.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
```

```
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)
read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`maaslin_reorder_data` *Reorder the abundance data and metadata.*

---

## Description

Reorder the abundance data and metadata to ensure samples are rows and remove any samples without abundances or metadata.

## Usage

```
maaslin_reorder_data(data,
                     metadata,
                     feature_specific_covariate = NULL,
                     unscaled_abundance = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A data frame of feature abundances or read counts. It should be formatted with features as columns and samples as rows (or the transpose). The column and row names should be the feature names and sample names respectively. |
| `metadata` | A data frame of per-sample metadata. It should be formatted with variables as columns and samples as rows (or the transpose). The column and row names should be the variable names and sample names respectively. |
| `feature_specific_covariate` | A table of feature-specific covariates or a filepath to a tab-delimited file with feature-specific covariates. It should be formatted with features as columns and samples as rows (or the transpose). The row names and column names should be the same as those of the `input_data`: the column and row names should be the feature names and sample names respectively. Typically, this table should be generated by ‘preprocess_mgx_mtx‘ or ‘preprocess_taxa_mtx‘ first. |

unscaled_abundance
: A data frame with a single column of absolute abundances. The row names should match the names of the samples in input_data and input_metadata. When using spike-ins, the single column should have the same name as one of the features in input_data, and the unscaled_abundance should correspond to the absolute quantity of the spike-in. When using total abundance scaling, the single column should have the name 'total', and the unscaled_abundance should correspond to the total abundance of each sample.

## Value

A list containing the following items:

**(1)** data: A data frame of feature abundances.

**(2)** metadata: A data frame of metadata.

**(3)** feature_specific_covariate: A data frame of feature specific covariates.

**(4)** unscaled_abundance: A data frame of unscaled abundances.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
```

```
        age + reads',
        plot_summary_plot = FALSE,
        plot_associations = FALSE)
    read_data_list <- maaslin3::maaslin_read_data(
        taxa_table,
        metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
        taxa_table,
        metadata)

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_transform` *Transform abundance data for MaAsLin 3 modeling.*

---

## Description

Transform the abundance data according to the `transform` parameter.

## Usage

```
maaslin_transform(filtered_data,
                  output,
                  transform = 'LOG')
```

## Arguments

| | |
|---|---|
| `filtered_data` | A data frame of filtered feature abundances. It should be formatted with features as columns and samples as rows. The column and row names should be the feature names and sample names respectively. |
| `output` | The output folder to write results. |
| `transform` | The transformation to apply to the features after normalization and before analysis. The option `LOG` (base 2) is recommended, but `PLOG` (pseudo-log) and `NONE` can also be used. |

## Value

A dataframe of transformed features (features are columns; samples are rows).

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                                output = 'output')

transformed_data = maaslin3::maaslin_transform(filtered_data,
```

```
                                    output = 'output')

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_write_results` *Write the results from a MaAsLin 3 run.*

---

## Description

Write the results from a MaAsLin 3 run to the output folder as a TSV.

## Usage

```
maaslin_write_results(output,
                      fit_data_abundance,
                      fit_data_prevalence,
                      random_effects_formula = NULL,
                      max_significance = 0.1,
                      save_models = FALSE)
```

## Arguments

| | |
|---|---|
| `output` | The output folder to write results. |
| `fit_data_abundance` | The abundance outputs of `maaslin_fit`. |
| `fit_data_prevalence` | The prevalence outputs of `maaslin_fit`. |
| `random_effects_formula` | A formula in `lme4` format as from `maaslin_check_formula`. |
| `max_significance` | The FDR corrected q-value threshold for significance used in selecting which associations to write as significant and to plot. |
| `save_models` | Whether to return the fit models and save them to an RData file. |

## Value

Results will be written to the `all_results.tsv` and `significant_results.tsv` files in the folder `output`. The file `all_results.tsv` will contain all results in the `fit_data_abundance` and `fit_data_prevalence` items of the input list (with 'linear' and 'logistic' replaced by 'abundance' and 'prevalence' in the `model` column). The file `significant_results.tsv` will contain all results with joint or individual q-values below the 'max_significance' parameter. No value is returned.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
    taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
    "maaslin3")
    taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

    # Read metadata table
    metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
    "maaslin3")
    metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

    metadata$diagnosis <-
        factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
    metadata$dysbiosis_state <-
        factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
        'dysbiosis_CD'))
    metadata$antibiotics <-
        factor(metadata$antibiotics, levels = c('No', 'Yes'))

    #Run MaAsLin3
    maaslin3::maaslin_log_arguments(
        input_data = taxa_table,
        input_metadata = metadata,
        output = 'output',
        formula = '~ diagnosis + dysbiosis_state + antibiotics +
        age + reads',
        plot_summary_plot = FALSE,
        plot_associations = FALSE)

    read_data_list <- maaslin3::maaslin_read_data(
        taxa_table,
        metadata)
        read_data_list <- maaslin3::maaslin_reorder_data(
        read_data_list$data,
        read_data_list$metadata)

    data <- read_data_list$data
    metadata <- read_data_list$metadata

    formulas <- maaslin3::maaslin_check_formula(
        data,
        metadata,
        input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
        age + reads')

    formula <- formulas$formula
    random_effects_formula <- formulas$random_effects_formula

    normalized_data = maaslin3::maaslin_normalize(data,
                                                  output = 'output')

    filtered_data = maaslin3::maaslin_filter(normalized_data,
```

```
                                                output = 'output')

    transformed_data = maaslin3::maaslin_transform(filtered_data,
                                                output = 'output')

    standardized_metadata = maaslin3::maaslin_process_metadata(
        metadata,
        formula = formula)

    maaslin_results = maaslin3::maaslin_fit(
        filtered_data,
        transformed_data,
        standardized_metadata,
        formula,
        random_effects_formula,
        warn_prevalence = FALSE)

    maaslin3::maaslin_write_results(
        output = 'output',
        maaslin_results$fit_data_abundance,
        maaslin_results$fit_data_prevalence,
        random_effects_formula)

    unlink('output', recursive=TRUE)
    logging::logReset()
```

---

`maaslin_write_results_lefse_format` *Write the results from a MaAsLin 3 run in LEfSe format.*

---

## Description

Write the results from a MaAsLin 3 run to the output folder in LEfSe format.

## Usage

```
maaslin_write_results_lefse_format(output,
                                   fit_data_abundance,
                                   fit_data_prevalence)
```

## Arguments

| | |
|---|---|
| `output` | The output folder to write results. |
| `fit_data_abundance` | The abundance outputs of `maaslin_fit`. |
| `fit_data_prevalence` | The prevalence outputs of `maaslin_fit`. |

## Value

Results will be written to the `lefse_style_results_abundance.res` file in the folder `output`. No value is returned.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
# Read features table
taxa_table_name <- system.file("extdata", "HMP2_taxonomy.tsv", package =
"maaslin3")
taxa_table <- read.csv(taxa_table_name, sep = '\t', row.names = 1)

# Read metadata table
metadata_name <- system.file("extdata", "HMP2_metadata.tsv", package =
"maaslin3")
metadata <- read.csv(metadata_name, sep = '\t', row.names = 1)

metadata$diagnosis <-
    factor(metadata$diagnosis, levels = c('nonIBD', 'UC', 'CD'))
metadata$dysbiosis_state <-
    factor(metadata$dysbiosis_state, levels = c('none', 'dysbiosis_UC',
    'dysbiosis_CD'))
metadata$antibiotics <-
    factor(metadata$antibiotics, levels = c('No', 'Yes'))

#Run MaAsLin3
maaslin3::maaslin_log_arguments(
    input_data = taxa_table,
    input_metadata = metadata,
    output = 'output',
    formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads',
    plot_summary_plot = FALSE,
    plot_associations = FALSE)

read_data_list <- maaslin3::maaslin_read_data(
    taxa_table,
    metadata)
    read_data_list <- maaslin3::maaslin_reorder_data(
    read_data_list$data,
    read_data_list$metadata)

data <- read_data_list$data
metadata <- read_data_list$metadata

formulas <- maaslin3::maaslin_check_formula(
```

```
    data,
    metadata,
    input_formula = '~ diagnosis + dysbiosis_state + antibiotics +
    age + reads')

formula <- formulas$formula
random_effects_formula <- formulas$random_effects_formula

normalized_data = maaslin3::maaslin_normalize(data,
                                output = 'output')

filtered_data = maaslin3::maaslin_filter(normalized_data,
                                output = 'output')

transformed_data = maaslin3::maaslin_transform(filtered_data,
                                output = 'output')

standardized_metadata = maaslin3::maaslin_process_metadata(
    metadata,
    formula = formula)

maaslin_results = maaslin3::maaslin_fit(
    filtered_data,
    transformed_data,
    standardized_metadata,
    formula,
    random_effects_formula,
    warn_prevalence = FALSE)

maaslin3::maaslin_write_results_lefse_format(
    output = 'output',
    maaslin_results$fit_data_abundance,
    maaslin_results$fit_data_prevalence)

unlink('output', recursive=TRUE)
logging::logReset()
```

---

`preprocess_dna_mtx` *Pre-process the DNA covariates for metatranscriptomics*

---

## Description

Pre-process the DNA covariates for metatranscriptomics by total-sum-scaling DNA abundances per sample and then, for each sample in each feature:

1. Log 2 transforming the DNA abundance if the DNA abundance is >=0
2. Setting the DNA abundance to `log2([minimum non-zero relative abundance in the dataset] / 2)` if the corresponding RNA abundance is non-zero but the DNA abundance is zero
3. Setting the DNA abundance to NA if both are zero

When the DNA is present, the RNA data can be modeled as usual in MaAsLin 3 with `log2(DNA)` as a covariate. When the DNA is not present, if the RNA is present, we assume the DNA was missed due to finite read depth, so the DNA abundance is imputed with a small pseudo-count. When neither the DNA nor RNA is present, we assume the gene/microbe was not in the sample and therefore no information about the transcription level can be obtained. Setting the DNA covariate to NA has the effect of dropping the sample when fitting the relevant feature's model in MaAsLin 3. Unlike most MaAsLin functions that will infer the samples from the row names and column names, **the** `rna_table` **must be formated as samples (rows) by features (columns)**.

## Usage

```
preprocess_dna_mtx(dna_table, rna_table)
```

## Arguments

| | |
|---|---|
| `dna_table` | The samples (rows) by features (columns) data frame of DNA abundances to preprocess. These can be relative abundances or counts. |
| `rna_table` | The samples (rows) by features (columns) data frame of RNA to preprocess. These can be relative abundances or counts. |

## Value

A list containing the following named items:

1. `dna_table`: The table of log2 transformed DNA relative abundances with NAs for any feature-sample pairs for which both the DNA and RNA abundances were 0.
2. `rna_table`: The table of total sum scaled RNA abundances. These are not log2 transformed.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
mgx_in <- data.frame('a' = c(1, 2, 0, 4, 5),
                     'b' = c(2, 3, 4, 5, 6),
                     'c' = c(3, 4, 5, 6, 0))
rownames(mgx_in) <- paste0("sample", c(1:5))

mtx_in <- data.frame('a' = c(1, 2, 3, 4, 5),
                     'b' = c(2, 3, 4, 5, 0),
                     'c' = c(3, 4, 5, 6, 0))
rownames(mtx_in) <- paste0("sample", c(1:5))

preprocess_out <- preprocess_dna_mtx(mgx_in, mtx_in)
```

`preprocess_taxa_mtx` *Pre-process the taxa covariates for metatranscriptomics*

## Description

Pre-process the taxa covariates for metatranscriptomics by total-sum-scaling the taxa, matching the taxa to the RNAs coming from those taxa, and then, for each sample in each feature:

1. Log 2 transforming the taxon abundance if the taxon abundance is >=0
2. Setting the taxon abundance to `log2([minimum non-zero relative abundance in the dataset] / 2)` if any of the corresponding RNA abundances are non-zero but the taxon abundance is zero
3. Setting the taxon abundance to NA if both are zero

When the taxon is present, the RNA data can be modeled as usual in MaAsLin 3 with `log2(taxon)` as a covariate. When the taxon is not present, if any of its RNA is present, we assume the taxon was missed due to finite read depth, so the taxon abundance is imputed with a small pseudo-count. When neither the taxon nor RNA is present, we assume the gene/microbe was not in the sample and therefore no information about the transcription level can be obtained. Setting the taxon covariate to NA has the effect of dropping the sample when fitting the relevant feature's model in MaAsLin 3. Unlike most MaAsLin functions that will infer the samples from the row names and column names, **the** `rna_table` **must be formated as samples (rows) by features (columns)**.

## Usage

```
preprocess_taxa_mtx(taxa_table, rna_table, rna_per_taxon)
```

## Arguments

| | |
|---|---|
| `taxa_table` | The samples (rows) by features (columns) data frame of taxon abundances to preprocess. These can be relative abundances or counts. |
| `rna_table` | The samples (rows) by features (columns) data frame of RNA to preprocess. These can be relative abundances or counts. |
| `rna_per_taxon` | A dataframe with the columns 'RNA' and 'taxon' with one row per 'RNA' column found in 'rna_table' giving both the 'RNA' column and which 'taxon' column it corresponds to in 'taxa_table'. |

## Value

A list containing the following named items:

1. `dna_table`: The table of log2 transformed taxon relative abundances with NAs for any feature-sample pairs for which both the taxon and RNA abundances were 0.
2. `rna_table`: The table of total sum scaled RNA abundances. These are not log2 transformed.

## Author(s)

William Nickols<willnickols@g.harvard.edu>,
Jacob Nearing<nearing@broadinstitute.org>,
Maintainers: Lauren McIver<lauren.j.mciver@gmail.com>,

## Examples

```
taxa_in <- data.frame('tax1' = c(1, 2, 0, 4, 5),
                'tax2' = c(2, 3, 4, 5, 6), check.names = FALSE)
rownames(taxa_in) <- paste0("sample", c(1:5))

mtx_in <- data.frame('a' = c(1, 2, 3, 4, 5),
                     'b' = c(2, 3, 4, 5, 0),
                     'c' = c(3, 4, 5, 6, 0), check.names = FALSE)
rownames(mtx_in) <- paste0("sample", c(1:5))

rna_per_taxon <- data.frame(RNA = c('a', 'b', 'c'),
                            taxon = c('tax1', 'tax1', 'tax2'))

preprocess_out <- preprocess_taxa_mtx(taxa_in, mtx_in, rna_per_taxon)
```

# Index