Package: readyomics (via r-universe)

May 17, 2026

Type Package

Title Ready-to-Use Omics Formatting, Analysis, and Visualization Pipeline

Version 0.2.0

Description Provides a flexible and streamlined pipeline for formatting, analyzing, and visualizing omics data, regardless of omics type (e.g. transcriptomics, proteomics, metabolomics). The package includes tools for shaping input data into analysis-ready structures, fitting linear or mixed-effect models, extracting key contrasts, and generating a rich variety of ready-to-use publication-quality plots. Designed for transparency and reproducibility across a wide range of study designs, with customizable components for statistical modeling.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports dplyr, future, future.apply, ggplot2, lme4, lmerTest, methods, permute, rlang, stats, stringr, tidyr, utils, vegan, zCompositions

Suggests ape, data.table, ggrepel, ggridges, IHW, imputeLCMD, knitr, phyloseq, progressr, purrr, qvalue, rmarkdown, ropls, scales, testthat (>= 3.0.0)

Roxygen list(markdown = TRUE)

BugReports https://github.com/lmartinezgili/readyomics/issues

URL https://lmartinezgili.github.io/readyomics/, https://github.com/lmartinezgili/readyomics

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs cmake make libicu-dev

Repository https://lmartinezgili.r-universe.dev

Date/Publication 2025-10-19 12:01:07 UTC

RemoteUrl https://github.com/lmartinezgili/readyomics

RemoteRef HEAD

RemoteSha 6cc9cac9cdffdd3385824a39402f59b89bbd4f7c

Contents

`add_feat_name` *Append feature names to a* dana *object*

Description

Adds a `feat_name` column to the dana object to map `feat_id` to original labels.

Usage

```
add_feat_name(dana_obj, feat_names)
```

Arguments

`dana_obj`	A dana object returned by `dana()`.
`feat_names`	A data frame mapping `feat_id` to `feat_name`. Must contain columns `"feat_id"` and `"feat_name"`.

Value

A modified version of `dana_obj`, with a `feat_name` column added to applicable components.

See Also

`dana()` for fitting differential analysis models on omics datasets.

Examples

```
set.seed(123)
mock_X <- matrix(rnorm(20 * 5), nrow = 20)
colnames(mock_X) <- paste0("feat_", seq_len(5))
rownames(mock_X) <- paste0("sample_", seq_len(20))

mock_names <- data.frame(
  feat_id = paste0("feat_", seq_len(5)),
  feat_name = c(
    "Glucose",
    "Lactic acid",
    "Citric acid",
    "Palmitic acid",
    "Cholesterol"
  ),
  stringsAsFactors = FALSE
)

sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 10)),
  time = factor(rep(c("T1", "T2"), times = 10)),
  subject_id = factor(rep(seq_len(10), each = 2)),
  stringsAsFactors = FALSE
)
rownames(sample_data) <- sample_data$sample_id

fit_df <- data.frame(
  feat_id = rep(colnames(mock_X), each = 2),
  Coefficient = rep(c("(Intercept)", "groupB"), 5),
  Estimate = rnorm(10),
  `Pr(>|t|)` = runif(10),
  padj = runif(10),
  stringsAsFactors = FALSE
)

# Mock dana object
dana_obj <- list(
  X = mock_X,
  sdata = sample_data,
  formula_rhs = ~ group,
  fit = fit_df,
  lrt = data.frame(),
  ranef = data.frame()
)
class(dana_obj) <- "dana"

# Add fearure labels
dana_obj <- dana_obj |>
  add_feat_name(mock_names)
```

add_taxa *Add taxonomic information to* dana *object*

Description

Appends features taxonomy to the dana object tables.

Usage

```
add_taxa(
  dana_obj,
  taxa_table,
 taxa_rank = c("asv", "substrain", "strain", "species", "genus", "family", "order",
    "class", "phylum", "domain")
)
```

Arguments

dana_obj	A dana object returned by dana().
taxa_table	A taxonomy table data.frame with taxonomy ranks in columns and row names corresponding to feat_ids in dana object.
taxa_rank	A character string specifying the taxonomy level of input features. Accepts one of: "asv", "substrain", "strain", "species", "genus", "family", "order", "class", "phylum", or "domain".

Details

- If taxa_rank = "asv", a taxon_name is constructed by pasting the ASV ID to the species (if available) or genus name.
- For other ranks, taxon_name is taken directly from the corresponding column in taxa_table.
- All higher-level taxonomy ranks available in taxa_table are also appended.

Value

A modified version of dana_obj, with taxonomy information added to relevant tables.

See Also

dana() for fitting differential analysis models on omics datasets.

Examples

```
set.seed(123)
mock_X <- matrix(rnorm(20 * 5), nrow = 20)
colnames(mock_X) <- paste0("feat_", seq_len(5))
rownames(mock_X) <- paste0("sample_", seq_len(20))
```

```
mock_taxa <- data.frame(
  Domain = rep("Bacteria", 5),
  Phylum = c("Firmicutes", "Bacteroidota", "Proteobacteria",
             "Actinobacteriota", "Firmicutes"),
  Class = c("Bacilli", "Bacteroidia", "Gammaproteobacteria",
            "Actinobacteria", "Clostridia"),
  Order = c("Lactobacillales", "Bacteroidales", "Enterobacterales",
            "Bifidobacteriales", "Clostridiales"),
  Family = c("Lactobacillaceae", "Bacteroidaceae", "Enterobacteriaceae",
             "Bifidobacteriaceae", "Clostridiaceae"),
  Genus = c("Lactobacillus", "Bacteroides", "Escherichia",
            "Bifidobacterium", "Clostridium"),
  Species = c("acidophilus", "fragilis", "coli", "longum", "butyricum"),
  row.names = paste0("feat_", seq_len(5)),
  stringsAsFactors = FALSE
)

sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 10)),
  time = factor(rep(c("T1", "T2"), times = 10)),
  subject_id = factor(rep(seq_len(10), each = 2)),
  stringsAsFactors = FALSE
)
rownames(sample_data) <- sample_data$sample_id

fit_df <- data.frame(
  feat_id = rep(colnames(mock_X), each = 2),
  Coefficient = rep(c("(Intercept)", "groupB"), 5),
  Estimate = rnorm(10),
  `Pr(>|t|)` = runif(10),
  padj = runif(10),
  stringsAsFactors = FALSE
)

# Mock dana object
dana_obj <- list(
  X = mock_X,
  sdata = sample_data,
  formula_rhs = ~ group,
  fit = fit_df,
  lrt = data.frame(),
  ranef = data.frame()
)
class(dana_obj) <- "dana"

# Add taxonomy
dana_obj <- dana_obj |>
  add_taxa(mock_taxa, taxa_rank = "genus")
```

adjust_pval *Adjust P-values in a* dana *object*

Description

Applies multiple testing correction to P-values from differential analysis results returned by the dana() function. Supports multiple adjustment methods and both coefficient and likelihood ratio test (LRT) P-values.

Usage

```
adjust_pval(
  dana_obj,
  padj_by = c("all", "terms"),
  padj_method = NULL,
  padj_method_LRT = NULL,
  ihw_covar = NULL,
  ihw_covar_id = NULL,
  ihw_args = list(),
  storey_args = list(),
  verbose = TRUE
)
```

Arguments

dana_obj	A dana class object returned by the dana() function.
padj_by	Character string. Whether P-value adjustment should be done globally across all coefficients ("all") or separately for each coefficient term ("terms").
padj_method	Character vector of one or more methods for adjusting P-values from coefficient tests. Defaults to "BH".
padj_method_LRT	Character vector of one or more methods for adjusting P-values from LRT tests. Defaults to "BH". P-values from LRT tests will always be adjusted independently for each LRT term.
ihw_covar	Data frame containing covariable(s) used for IHW::ihw(). Must contain a "feat_id" column, matching dana object "feat_id" labels.
ihw_covar_id	Character string. Column name in ihw_covar used as covariates argument in IHW::ihw().
ihw_args	Named list. Additional arguments passed to IHW::ihw(). Do not provide covariates argument, as it will be obtained from ihw_covar.
storey_args	Named list. Additional arguments passed to qvalue::qvalue().
verbose	Logical. Whether to print informative messages. Defaults to TRUE.

Details

Available adjustment methods include: `"BH"`, `"bonferroni"`, `"BY"`, `"fdr"`, `"hochberg"`, `"holm"`, `"hommel"`, `"IHW"`, and `"storey"`.

Value

A modified dana object with new columns in the `$fit` and `$lrt` data frames for each adjusted P-value method applied (e.g. `padj_BH`, `padj_storey_group`).

See Also

- `dana()` for fitting differential analysis models on omics datasets.
- `IHW::ihw()` for inverted hypothesis weighting method details.
- `qvalue::qvalue()` for Storey's qvalue method details.

Examples

```
set.seed(123)
mock_X <- matrix(rnorm(20 * 5), nrow = 20)
colnames(mock_X) <- paste0("feat_", seq_len(5))
rownames(mock_X) <- paste0("sample_", seq_len(20))

sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 10)),
  time = factor(rep(c("T1", "T2"), times = 10)),
  subject_id = factor(rep(seq_len(10), each = 2)),
  stringsAsFactors = FALSE
)
rownames(sample_data) <- sample_data$sample_id

fit_df <- data.frame(
  feat_id = rep(colnames(mock_X), each = 2),
  Coefficient = rep(c("(Intercept)", "groupB"), 5),
  Estimate = rnorm(10),
  `Pr(>|t|)` = runif(10),
  stringsAsFactors = FALSE
)

# Mock dana object
dana_obj <- list(
  X = mock_X,
  sdata = sample_data,
  formula_rhs = ~ group,
  fit = fit_df,
  lrt = data.frame(),
  ranef = data.frame()
)
class(dana_obj) <- "dana"

# Add adjusted P-values
```

```
dana_obj <- dana_obj |>
  adjust_pval(padj_method = c("BH", "bonferroni"),
              padj_method_LRT = NULL,
              padj_by = "terms",
              verbose = FALSE)
```

`build_phyloseq` *Build phyloseq objects for all taxonomy ranks*

Description

Constructs a list of phyloseq objects from a feature matrix (`X`), sample data, taxonomy and (optionally) phylogenetic tree data.

Usage

```
build_phyloseq(
  X,
  sample_data,
  taxa_table = NULL,
  phylo_tree = NULL,
  taxa_in_rows,
  verbose = TRUE
)
```

Arguments

`X`	A numeric matrix of NGS features (e.g., ASVs), with samples in rows and features in columns (recommended) or vice versa.
`sample_data`	A `data.frame` containing sample data. Row names must match sample identifiers in `X`.
`taxa_table`	(Optional) A taxonomy table with row names corresponding to feature names in `X`, and taxonomic ranks as columns.
`phylo_tree`	(Optional) A phylogenetic tree.
`taxa_in_rows`	Logical. If `TRUE`, `X` is assumed to have taxa as rows and samples as columns.
`verbose`	Logical. If `TRUE`, diagnostic messages will be printed.

Details

Phyloseq objects for higher taxonomic ranks are also generated when `taxa_table` is provided. Higher rank taxa with labels matching "unclass" or "unknown" are excluded after aggregation.

If very long strings are detected as feature IDs in `X` matrix or `taxa_table`, (for example when actual DNA sequence is used as ID), it will issue a warning, as this could significantly slow down computation and increase memory usage.

Value

A named `list` of `phyloseq` objects and related output:

asv Phyloseq object with the raw feature counts (usually ASVs).

<tax_rank> Phyloseq objects of higher taxonomy ranks from `taxa_table`.

See Also

`phyloseq::phyloseq()` for further details on phyloseq objects.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
mock_X <- matrix(c(10, 0, 5, 3, 1, 7),
                 nrow = 2, byrow = TRUE,
                 dimnames = list(c("sample1", "sample2"),
                                 c("ASV1", "ASV2", "ASV3"))
                 )

mock_sample_data <- data.frame(sample_id = c("sample1", "sample2"),
                               group = c("A", "B"),
                               row.names = c("sample1", "sample2")
                               )

mock_taxa_table <- data.frame(Domain = c("Bacteria", "Bacteria", "Bacteria"),
                              Genus = c("GenusA", "GenusB", "Unknown"),
                              row.names = c("ASV1", "ASV2", "ASV3")
                              )

phyloseq_ready <- build_phyloseq(X = mock_X,
                                 sample_data = mock_sample_data,
                                 taxa_table = mock_taxa_table,
                                 taxa_in_rows = FALSE,
                                 verbose = FALSE)
}
```

---

dana *Differential analysis (dana)*

Description

Feature-wise `stats::lm()` or `lme4::lmer()` models of an omics data matrix. Supports likelihood ratio tests (LRT) and parallel computation.

Usage

```
dana(
  X,
  sample_data,
  formula_rhs,
  term_LRT = NULL,
  model_control = list(),
  platform = c("ms", "nmr", "ngs"),
  assay = NULL,
  verbose = TRUE
)
```

Arguments

X	A numeric matrix with samples in rows and features in columns. Sample IDs in row names must match the format from sample_id column in sample_data.
sample_data	A data frame containing sample-level data. Must have a sample_id column matching row names in X and sample_data.
formula_rhs	A one-sided formula (e.g., ~ group + (1|subject)). Must not contain a response variable.
term_LRT	Optional. Character vector of formula terms to test via LRT. Random effects must be written without parentheses (e.g., "1 | group").
model_control	Optional. List of control arguments passed to the model.
platform	Character string indicating the omics platform (e.g., "ms", "nmr", "ngs").
assay	Optional. Character string indicating the name of the platform assay (e.g., "lipidomics").
verbose	Logical. If TRUE, prints progress messages.

Details

Models are fit independently for each feature using stats::lm() or lmerTest::lmer(), depending on whether dana() detects random effects in formula_rhs. Feature-wise models can be evaluated in parallel using future::plan(), with optional progress updates via progressr::with_progress().

Value

An object of class "dana":

X Matched data matrix.

sdata Matched sample data.

fit Data frame of model coefficients and confidence intervals per feature.

lrt Likelihood ratio test results (if term_LRT is specified).

ranef Random effects variance components (if using mixed models).

errors A data frame logging any model fitting errors per feature.

See Also

stats::lm(), lme4::lmer(), lmerTest::lmer() parameters.

Examples

```
mock_X <- matrix(
  rnorm(50 * 10) +
    rep(c(rep(0, 25), rep(2, 25)), each = 10) * rep(1:10 %in% 1:3, each = 50),
  nrow = 50
)

rownames(mock_X) <- paste0("sample", 1:50)
colnames(mock_X) <- paste0("feat", 1:10)

sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 25)),
  subject = factor(rep(1:25, each = 2)),
  row.names = rownames(mock_X)
)

# Example with parallel computation setup (not run)
# future::plan(multisession)
# progressr::handlers(global = TRUE)
# progressr::with_progress({
  result <- dana(X = mock_X,
                 sample_data = sample_data,
                 formula_rhs = ~ group + (1 | subject),
                 term_LRT = c("group", "1 | subject"), # Multiple terms allowed
                 platform = "ms",
                 assay = "lipidomics",
                 verbose = FALSE
                 )
# })

# Modify `dana` object at once with pipes (not run)
# dana_obj <- dana_obj |> adjust_pval() |> add_feat_name() |> ready_plots()
```

---

mva *Multivariate analysis (PCA, PLS, OPLS)*

Description

Performs PCA, PLS, or OPLS using ropls and generates a formatted scores plot based on the first two components.

Usage

```
mva(
  X,
  sample_data,
  group_colour = NULL,
  group_shape = NULL,
  plot_title = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

X	A numeric matrix or data frame of features (e.g., metabolites, genes), with samples as rows and features as columns.
sample_data	A data.frame containing sample-level data. Row names must match the sample identifiers in X and must be also in a column named "sample_id".
group_colour	Optional. Character colname in sample_data used for point color mapping.
group_shape	Optional. Character colname in sample_data used for point shape mapping.
plot_title	Optional. Character string specifying the plot title.
verbose	Logical. If TRUE, displays progress messages.
...	Additional arguments passed to ropls::opls() (e.g.predI =, orthoI =).

Details

The analysis type depends on the ... arguments passed to ropls::opls().

Value

A named list with two elements:

ropls_obj The ropls::opls() object.

scores_plot A ggplot2::ggplot() object showing the scores plot.

See Also

ropls::opls() for details on the ropls::opls() output.

Examples

```
# PCA
if (requireNamespace("ropls", quietly = TRUE)) {
set.seed(123)
mock_X <- matrix(rnorm(40),
                 nrow = 10,
                 dimnames = list(paste0("sample", 1:10),
                                 paste0("feat", 1:4))
                 )
```

```
sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 5)),
  batch = factor(rep(1:2, times = 5)),
  row.names = rownames(mock_X),
  stringsAsFactors = FALSE
)

result <- mva(
  X = mock_X,
  sample_data = sample_data,
  group_colour = "group",
  group_shape = "batch",
  plot_title = "Test PCA Plot",
  predI = 2,  # PCA: set components
  verbose = FALSE
)

# PCA plot
result$scores_plot
}
```

---

permanova *PERMANOVA with flexible permutation control*

Description

Performs PERMANOVA (Permutational Multivariate Analysis of Variance). Supports both joint-term (default `vegan::adonis2()`) and single-term testing when `independent = TRUE`. Several distance methods, and fine-grained permutation control.

Usage

```
permanova(
  X,
  sample_data,
  formula_rhs,
  dist_control = list(method = "euclidean", diag = FALSE, upper = FALSE),
 perm_control = list(joint_terms = list(control = permute::how(blocks = NULL, nperm =
    999))),
  independent = TRUE,
  platform = c("ms", "nmr", "ngs"),
  assay = NULL,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

X	A processed matrix or data frame of features (samples in rows, features in columns).
sample_data	A data.frame containing sample-level data. Row names must match those in X.
formula_rhs	A one-sided formula (e.g., ~ group + age).
dist_control	A named list of arguments to control distance calculation. Must contain at least method. Defaults to "Euclidean" via stats::dist().
perm_control	A named list specifying permute::shuffleSet() parameters. By default, joint_terms parameters will be used, with same vegan::adonis2() defaults, unless variable-specific permutation settings are added as named list elements (e.g. perm_control = list(joint_terms = , age = , sex =)).
independent	Logical. If TRUE, a PERMANOVA test for each variable in formula_rhs is performed.
platform	A string specifying the omics platform ("ms", "nmr", "ngs"). Used for annotation.
assay	Optional. Character string giving the assay name for annotation (e.g., "lipidomics").
seed	Optional integer. If provided, sets the random seed for reproducible permutation results.
verbose	Logical. If TRUE, prints diagnostic messages.
...	Additional arguments passed to vegan::adonis2().

Details

- Supports both stats::dist() and vegan::vegdist() for distance matrix computation.
- Distance method must be specified in dist_control$method.
- Permutation design is controlled via the permute package using permute::shuffleSet().
- If seed is supplied, the same permutations will be used across runs for reproducibility.

Value

A named list with three elements:

X_dist A dist object.

perm_matrix_joint A matrix from permute::shuffleSet() joint_terms control.

permanova_joint A data.frame of PERMANOVA results using the full model.

permanova_indep A data.frame of a PERMANOVA results for each predictor, or NULL if independent = FALSE.

See Also

- stats::dist() and vegan::vegdist() for information on available distances.
- vegan::adonis2() and permute::shuffleSet() for control options and details.
- process_ngs() to pre-process and normalize an X NGS dataset.
- process_ms() to pre-process and normalize an X MS dataset.

Examples

```
# Mock data
X <- matrix(rnorm(40), nrow = 10,
            dimnames = list(paste0("sample", 1:10),
                            paste0("feat", 1:4)))
sample_data <- data.frame(
  sample_id = rownames(X),
  group = factor(rep(c("A", "B"), each = 5)),
  age = rep(20:29, length.out = 10),
  row.names = rownames(X),
  stringsAsFactors = FALSE
)

# Simple control structures
dist_control <- list(method = "euclidean")
perm_control <- list(
  joint_terms = list(control = permute::how(blocks = NULL, nperm = 9)),
  group = list(control = permute::how(blocks = NULL, nperm = 9)),
  age = list(control = permute::how(blocks = NULL, nperm = 9))
)

result <- permanova(
  X = X,
  sample_data = sample_data,
  formula_rhs = ~ group + age,
  dist_control = dist_control,
  perm_control = perm_control,
  independent = TRUE,
  platform = "ms",
  assay = "lipidomics",
  seed = 42,
  verbose = FALSE
)
```

`process_ms` *Process MS-like omics data*

Description

This function performs common preprocessing steps for mass spectrometry (MS)-like omics datasets, including QC sample removal, zero-to-NA conversion, feature prevalence filtering, transformation, and feature-wise value imputation.

Usage

```
process_ms(
  X,
  remove_ids = NULL,
```

```
  min_prev = 0.8,
  rename_feat = TRUE,
  transform = c("none", "log", "sqrt"),
  log_base_num = 10,
  impute = c("none", "min_val", "QRILC"),
  min_val_factor = 1,
  platform = c("ms", "nmr"),
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

X	A numeric data frame or matrix (samples in rows, features in columns).
remove_ids	A regex or character vector to filter out rows in X (e.g. QCs). Set to NULL to skip.
min_prev	Numeric between 0 and 1. Minimum non-missing prevalence threshold. Zeros are first converted to NA.
rename_feat	Logical. If TRUE, features will be renamed as "feat_n" and original labels stored.
transform	One of "none", "log", or "sqrt".
log_base_num	Numeric logarithm base. Required if transform = "log".
impute	One of "none", "min_val", or "QRILC". Note: imputeLCMD::impute.QRILC() requires log-transformed data. Log-transform will be forced internally regardless of transform = setting.
min_val_factor	Numeric >= 1. Scaling factor for min value imputation.
platform	whether data was generated by mass spectrometry ("ms") or nuclear magnetic resonance spectroscopy ("nmr"), the latter allowing negative values in the matrix.
seed	Optional integer. If provided, sets the random seed for reproducible imputeLCMD::imputeQRILC() permutation results.
verbose	Logical. Show messages about the processing steps.
...	Extra arguments passed to imputeLCMD::impute.QRILC().

Value

A list:

X_names Feature mapping original vs. new names.

X_processed Processed numeric matrix.

References

Lazar, C., Gatto, L., Ferro, M., Bruley, C., & Burger, T. (2016). Accounting for the multiple natures of missing values in label-free quantitative proteomics data sets to compare imputation strategies. *Journal of Proteome Research*, 15(4), 1116–1125. doi:10.1021/acs.jproteome.5b00981

Wei, R., Wang, J., Su, M., Jia, E., Chen, S., Chen, T., & Ni, Y. (2018). Missing value imputation approach for mass spectrometry-based metabolomics data. *Scientific Reports*, 8, 663. doi:10.1038/s41598017191200

See Also

`imputeLCMD::impute.QRILC()` for imputing missing values.

Examples

```
X <- matrix(sample(c(0:10), size = 80, replace = TRUE),
            nrow = 20, ncol = 4,
            dimnames = list(paste0("sample", 1:20),
                            paste0("feat", 1:4)))

result <- process_ms(X, verbose = FALSE) # Generates NA warning
```

`process_ngs` *Process next generation sequencing data*

Description

This function performs quality control, filtering, normalization, and transformation of sequencing data raw counts. It can also build `phyloseq` objects for downstream ecological analyses, and optionally returns intermediate processing steps.

Usage

```
process_ngs(
  X,
  sample_data,
  taxa_table = NULL,
  phylo_tree = NULL,
  remove_ids = NULL,
  min_reads = 500,
  min_prev = 0.1,
  normalise = c("load", "TSS", "none"),
  load_colname = NULL,
  min_load = 10000,
  transform = c("clr", "log", "none"),
 impute_control = list(method = "GBM", output = "p-counts", z.delete = FALSE, z.warning
    = 1, suppress.print = TRUE),
  raw_phyloseq = TRUE,
  eco_phyloseq = TRUE,
  return_all = FALSE,
  verbose = TRUE
)
```

Arguments

X	A numeric matrix or data frame of raw counts with samples as rows and features (e.g., taxa) as columns. Row names must be sample IDs.
sample_data	A data frame containing sample-level data. Must include a column named sample_id with matching row names with X.
taxa_table	Optional. Taxonomy annotation table to build phyloseq objects. Row names must match column names of X.
phylo_tree	Optional. Phylogenetic tree to add to phyloseq objects.
remove_ids	A regex or character vector to filter rows in X. Set to NULL to skip.
min_reads	Numeric. Minimum number of total reads required per sample. Default is 500.
min_prev	Numeric between 0 and 1. Minimum feature prevalence threshold. Default is 0.1 (i.e., feature must be present in >= 10 % of samples).
normalise	Normalization method. One of "load" (microbial load data), "TSS" (total sum scaling), or "none".
load_colname	Column name in sample_data containing microbial load values. Required if normalise = "load".
min_load	Numeric. Default is 1e4. Warns if any microbial load value < min_load.
transform	Transformation method. One of "clr" (centered log-ratio with zero imputation), "log" (pseudo-log using log1p()), or "none". Note: When using "clr", zero values are imputed using zCompositions::cmultRepl().
impute_control	A named list of arguments to be passed to zCompositions::cmultRepl().
raw_phyloseq	Logical. If TRUE, constructs a phyloseq object with the table of raw counts (filtered failed runs if needed). Default is TRUE.
eco_phyloseq	Logical. If TRUE, constructs a phyloseq object with the ecosystem abundances (i.e. after normalise = "load"). Default is TRUE.
return_all	Logical. If TRUE, additional intermediate data matrices (X_matched, X_norm, X_prev) are included in the output. Default is FALSE.
verbose	Logical. If TRUE, prints progress messages during execution. Default is TRUE.

Details

- Zeros are imputed with zCompositions::cmultRepl() before CLR transformation.
- QC or other samples are removed if remove_ids is specified.
- Sample IDs in X and sample_data row names are matched and aligned.
- Can generate both a phyloseq_raw phyloseq object containing raw counts and a phyloseq_eco object with ecosystem counts, if a load_colname column from sample_data is provided to normalize the counts by microbial load (recommended best practice).

Value

A named list containing:

X_processed Matrix of processed feature counts after filtering, normalization, and transformation.

`sdata_final` Matched and filtered `sample_data` corresponding to retained samples.

`phyloseq_raw` `phyloseq` object created from raw filtered data. `NULL` if `raw_phyloseq = FALSE`.

`phyloseq_eco` `phyloseq` object from ecosystem abundance data. `NULL` if `eco_phyloseq = FALSE` or `normalise != "load"`.

`X_matched` (Optional) Matched and filtered count matrix, pre-normalization. Returned only if `return_all = TRUE`.

`X_norm` (Optional) Normalized count matrix. Returned only if `return_all = TRUE`.

`X_prev` (Optional) Prevalence-filtered matrix, pre-transformation. Returned only if `return_all = TRUE`.

References

#' McMurdie, P. J., & Holmes, S. (2013). phyloseq: An R package for reproducible interactive analysis and graphics of microbiome census data. *PLoS ONE*, 8(4), e61217. doi:10.1371/journal.pone.0061217

Martín-Fernández, J. A., Hron, K., Templ, M., Filzmoser, P., & Palarea-Albaladejo, J. (2015). Bayesian-multiplicative treatment of count zeros in compositional data sets. *Statistical Modelling*, 15(2), 134–158. doi:10.1177/1471082X14535524

Palarea-Albaladejo, J., & Martín-Fernández, J. A. (2015). zCompositions—R package for multivariate imputation of left-censored data under a compositional approach. *Chemometrics and Intelligent Laboratory Systems*, 143, 85–96. doi:10.1016/j.chemolab.2015.02.019

Gloor, G. B., Macklaim, J. M., Pawlowsky-Glahn, V., & Egozcue, J. J. (2017). Microbiome datasets are compositional: And this is not optional. *Frontiers in Microbiology*, 8, 2224. doi:10.3389/fmicb.2017.02224

Vandeputte, D., Kathagen, G., D'hoe, K., Vieira-Silva, S., Valles-Colomer, M., Sabino, J., Wang, J., Tito, R. Y., De Commer, L., Darzi, Y., Vermeire, S., Falony, G., & Raes, J. (2017). Quantitative microbiome profiling links gut community variation to microbial load. *Nature*, 551(7681), 507–511. doi:10.1038/nature24460

See Also

- `build_phyloseq()`
- `zCompositions::cmultRepl()`

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
mock_X <- matrix(sample(0:1000, 25, replace = TRUE),
                 nrow = 5,
                 dimnames = list(paste0("sample", 1:5),
                 paste0("ASV", 1:5))
                 )

mock_sample_data <- data.frame(
  sample_id = paste0("sample", 1:5),
  load = c(1e5, 2e5, 1e4, 5e4, 1.5e5),
  condition = factor(rep(c("A", "B"), length.out = 5)),
```

```
  row.names = paste0("sample", 1:5)
  )

mock_taxa_table <- data.frame(
  Kingdom = rep("Bacteria", 5),
  Genus = paste0("Genus", 1:5),
  row.names = paste0("ASV", 1:5)
  )

result <- process_ngs(
  X = mock_X,
  sample_data = mock_sample_data,
  taxa_table = mock_taxa_table,
  normalise = "load",
  load_colname = "load",
  transform = "none",
  verbose = FALSE
  )
}
```

`ready_plots` *Generate plots from a differential analysis (*dana*) object*

Description

This function produces a range of coefficient- and feature-level plots from a dana object for a given model term of interest. It supports both main effect and interaction terms, and can visualize significant results from either `fit` or `lrt` P values.

Usage

```
ready_plots(
  dana_obj,
  term_name,
  pval_match,
  alpha = 0.1,
  add_interactions = TRUE,
  add_labels = TRUE,
  plot_coeff = TRUE,
  plot_feat = TRUE,
  plot_ranef = FALSE,
  X_colnames = NULL,
  sdata_var = NULL,
  group_colours = NULL,
  paired_id = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

dana_obj	A dana object returned by dana(), containing model results.
term_name	The name of the model term to plot (e.g., "group" or "group:time").
pval_match	Regex pattern to match the desired P value column in the results.
alpha	Numeric. Significance threshold to consider features for plotting. Default 0.1.
add_interactions	Logical. Whether to include interaction terms related to term_name.
add_labels	Logical. Whether to add custom feature labels in plots. A "feat_name" or "taxon_name" column must be in the dana object. See add_taxa() and add_feat_name().
plot_coeff	Logical. Whether to generate coefficient-level plots. Will generate volcano, heatmap and dot plots.
plot_feat	Logical. Whether to generate feature-level plots for a specific variable in sample_data.
plot_ranef	Logical. Whether to generate random effect variance plots. Only for mixed-effects models.
X_colnames	Optional. Character vector specifying which features from X to plot. If NULL and plot_feat = TRUE (the default), top 10 features based on P value are selected.
sdata_var	Character. A column in dana_obj$sdata used for feature-level plots when plot_feat = TRUE.
group_colours	Optional named vector of colours for sdata_var groups to be passed as values argument to ggplot2::scale_fill_manual().
paired_id	Optional. Column name in sdata specifying sample pairing (e.g., subject_id).
verbose	Logical. Whether to display messages during processing.
...	Additional ggplot2::theme() arguments passed to internal plotting helpers (e.g., font sizes).

Details

When add_interactions = TRUE, the function shows fit coefficients that match significant main and interaction terms.

If no significant features are found under the specified alpha significance threshold, the function will abort.

Value

A named list of ggplot objects stored in dana_obj$plots. These may include:

- coeff_volcano, coeff_heatmap, coeff_point
- feat_scatter, feat_boxplot, feat_violin, feat_ridge
- ranef_all

See Also

- dana() for fitting differential analysis models on omics datasets.
- add_taxa() and add_feat_name() for adding feature labels to dana object.
- ggplot2::ggplot() and ggplot2::theme() to further customise plots.

Examples

```
set.seed(123)
mock_X <- matrix(rnorm(20 * 5), nrow = 20)
colnames(mock_X) <- paste0("feat_", seq_len(5))
rownames(mock_X) <- paste0("sample_", seq_len(20))

sample_data <- data.frame(
  sample_id = rownames(mock_X),
  group = factor(rep(c("A", "B"), each = 10)),
  time = factor(rep(c("T1", "T2"), times = 10)),
  subject_id = factor(rep(seq_len(10), each = 2)),
  stringsAsFactors = FALSE
)
rownames(sample_data) <- sample_data$sample_id

fit_df <- data.frame(
  feat_id = rep(colnames(mock_X), each = 2),
  Coefficient = rep(c("(Intercept)", "groupB"), 5),
  Estimate = rnorm(10),
  `Pr(>|t|)` = runif(10),
  padj = runif(10),
  stringsAsFactors = FALSE
)

# Mock dana object
dana_obj <- list(
  X = mock_X,
  sdata = sample_data,
  formula_rhs = ~ group,
  fit = fit_df,
  lrt = data.frame(),  #' empty but valid
  ranef = data.frame() #' empty but valid
)
class(dana_obj) <- "dana"

dana_obj <- dana_obj |>
ready_plots(
  term_name = "group",
  pval_match = "padj",
  alpha = 0.5,
  add_labels = FALSE,
  plot_coeff = TRUE,
  plot_feat = TRUE,
  plot_ranef = FALSE,
  sdata_var = "group",
  verbose = FALSE
)

# Visualize generated plots
dana_obj$plots
```

Index