

Package: OmicFlow (via r-universe)

May 28, 2026

Title Fast and Efficient (Automated) Analysis of Sparse Omics Data

Version 1.6.0

Date 2026-05-19

Description A generalised data structure for fast and efficient loading and data munching of sparse omics data. The 'OmicFlow' requires an up-front validated metadata template from the user, which serves as a guide to connect all the pieces together by aligning them into a single object that is defined as an 'omics' class. Once this unified structure is established, users can perform manual subsetting, visualisation, and statistical analysis, or leverage the automated 'autoFlow' method to generate a comprehensive report.

License MIT + file LICENSE

URL <https://github.com/agusinac/OmicFlow>

BugReports <https://github.com/agusinac/OmicFlow/issues>

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.3.0), R6, data.table, Matrix

Imports ape, ggpubr, ggrepel, ggplot2, jsonlite, jsonvalidate, magrittr, methods, patchwork, RColorBrewer, rhdf5, rstatix, Rcpp (>= 0.12.6), RcppParallel (>= 4.3.20), stats, tools, utils, vegan, yyjsonr, cli

Suggests DT, downloadthis, rmarkdown, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppParallel, RcppArmadillo

Config/pak/sysreqs cmake make libicu-dev libssl-dev libnode-dev

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

Date/Publication 2026-05-19 11:37:59 UTC

RemoteUrl <https://github.com/agusinac/omicflow>

RemoteRef HEAD

RemoteSha 07400f9adbd2455fc5975c68df67154fb219561e

Contents

bray	2
canberra	3
colormap	5
column_exists	5
composition_plot	6
cosine	8
diversity	9
diversity_plot	10
euclidean	12
hill_taxa	13
jaccard	14
jsd	15
manhattan	16
matrix_to_dtable	18
metagenomics	18
omics	22
ordination_plot	42
pairwise_adonis	44
pairwise_anosim	45
plot_pairwise_stats	46
proteomics	48
unifrac	49
volcano_plot	51
Index	53

bray	<i>Compute Bray-Curtis Dissimilarity from a Dense or Sparse Matrix.</i>
------	---

Description

Calculates the Bray-Curtis dissimilarity of a Matrix pairwise for each column.

Usage

```
bray(x, weighted = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The Bray-Curtis dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = \frac{\sum_i^n |A_i - B_i|}{\sum_i^n (A_i + B_i)}$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. When weighted is set to FALSE, counts are replaced by presence/absence data.

Value

A column x column `dist` object.

References

Bray, J.R. & Curtis, J.T. (1957) An Ordination of the Upland Forest Communities of Southern Wisconsin. *Ecological Monographs*, 27(4), 325–349.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

bray(taxa$countData)
```

canberra	<i>Compute Canberra Dissimilarity from a from a Dense or Sparse Matrix.</i>
----------	---

Description

Calculates the Canberra dissimilarity of a Matrix pairwise for each column.

Usage

```
canberra(x, weighted = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The Canberra dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = \frac{1/NZ^n \sum_i |A_i - B_i|}{\sum_i |A_i| + |B_i|}$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. NZ are the number of non-zero entries. When `weighted` is set to `FALSE`, counts are replaced by presence/absence data.

Value

A column x column [dist](#) object.

References

Lance, G.N. & Williams, W.T. (1967) Mixed-data classificatory programs. I. Agglomerative systems. *Australian Computer Journal*, 1(1), 15-20.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

canberra(taxa$countData)
```

colormap	<i>Color map of a variable</i>
----------	--------------------------------

Description

Creates an object of hexcode colors with names given a vector of characters. This function is built into the ordination method from the abstract class [omics](#) and inherited by other omics classes, such as; [metagenomics](#) and [proteomics](#).

Usage

```
colormap(data, col_name, Brewer.palID = "Set2")
```

Arguments

data	A data.frame or data.table .
col_name	A column name of a categorical variable.
Brewer.palID	A character name that exists in brewer.pal (Default: "Set2").

Value

A [setNames](#).

Examples

```
library("data.table")
dt <- data.table(
  "SAMPLE_ID" = c("sample_1", "sample_2", "sample_3"),
  "treatment" = c("healthy", "tumor", NA)
)

colors <- colormap(data = dt,
  col_name = "treatment")
```

column_exists	<i>Checks if column exists in table</i>
---------------	---

Description

Mainly used within [omics](#) and other functions to check if given column name does exist in the table and is not completely empty (containing NAs).

Usage

```
column_exists(column, table)
```

Arguments

column	A character of length 1.
table	A data.table or data.frame .

Value

A boolean value.

composition_plot	<i>Compositional plot</i>
------------------	---------------------------

Description

Creates a stacked barchart of features. It is possible to both show barcharts for each sample or group them by a categorical variable. The function is compatible with the class [omics](#) method `composition()`.

Usage

```
composition_plot(  
  data,  
  palette,  
  feature_rank,  
  title_name = NULL,  
  group_by = NULL  
)
```

Arguments

data	A data.frame or data.table .
palette	An object with names and hexcode or color names, see colormap .
feature_rank	A character variable of the feature column.
title_name	A character to set the ggtitle of the ggplot , (Default: NULL).
group_by	A character variable to aggregate the stacked bars by group (Default: NULL).

Value

A [ggplot2](#) object to be further modified

Examples

```

library("ggplot2")

# Create mock_data as data.frame (data.table is also supported)
mock_data <- data.frame(
  SAMPLE_ID = rep(paste0("Sample", 1:10), each = 5),
  Genus = rep(c("GenusA", "GenusB", "GenusC", "GenusD", "GenusE"), times = 10),
  value = c(
    0.1119, 0.1303, 0.0680, 0.5833, 0.1065, # Sample1
    0.2080, 0.1179, 0.0211, 0.4578, 0.1951, # Sample2
    0.4219, 0.1189, 0.2320, 0.1037, 0.1235, # Sample3
    0.4026, 0.0898, 0.1703, 0.1063, 0.2309, # Sample4
    0.1211, 0.0478, 0.5721, 0.1973, 0.0618, # Sample5
    0.2355, 0.0293, 0.2304, 0.1520, 0.3528, # Sample6
    0.2904, 0.0347, 0.3651, 0.0555, 0.2544, # Sample7
    0.4138, 0.0299, 0.0223, 0.4996, 0.0345, # Sample8
    0.4088, 0.0573, 0.0155, 0.2888, 0.2296, # Sample9
    0.4941, 0.0722, 0.2331, 0.1023, 0.0983 # Sample10
  ),
  Group = rep(c("Group1", "Group2", "Group1",
                "Group1", "Group2", "Group2",
                "Group1", "Group1", "Group1", "Group2"),
              each = 5)
)

# Create a colormap
mock_palette <- c(
  GenusA = "#1f77b4", # blue
  GenusB = "#ff7f0e", # orange
  GenusC = "#2ca02c", # green
  GenusD = "#d62728", # red
  GenusE = "#9467bd" # purple
)

# Optionally: Use OmicFlow::colormap()
mock_palette <- colormap(
  data = mock_data,
  col_name = "Genus",
  Brewer.palID = "RdYlBu"
)

composition_plot(
  data = mock_data,
  palette = mock_palette,
  feature_rank = "Genus",
  title_name = "Mock Genus Composition"
)

composition_plot(
  data = mock_data,
  palette = mock_palette,
  feature_rank = "Genus",

```

```

  title_name = "Mock Genus Composition by Group",
  group_by = "Group"
)

```

cosine

Compute Cosine Dissimilarity from a Dense or Sparse Matrix.

Description

Calculates the cosine dissimilarity of a Matrix pairwise for each column.

Usage

```
cosine(x, weighted = TRUE, threads = 1)
```

Arguments

x A [matrix](#), [sparseMatrix](#) or [Matrix](#).

weighted A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).

threads A wholenumber, the number of threads to use in [setThreadOptions](#) (default: 1).

Details

The cosine dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = 1 - \frac{\sum_i^n A_i B_i}{\sqrt{\sum_i^n A_i^2} \sqrt{\sum_i^n B_i^2}}$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. When weighted is set to FALSE, counts are replaced by presence/absence data.

Value

A column x column [dist](#) object.

References

Deza, M. M., & Deza, E. (2009). Encyclopedia of Distances. Springer Science & Business Media., 308.

Examples

```

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

```

```
taxa <- metagenomics$new(  
  metaData = metadata_file,  
  countData = counts_file,  
  featureData = features_file,  
  treeData = tree_file  
)  
  
taxa$feature_subset(Kingdom == "Bacteria")  
taxa$scale(method = "tss")  
  
cosine(taxa$countData)
```

diversity

Sparse implementation of Alpha Diversity Metrics

Description

Computes the alpha diversity based on Shannon index, simpson or invsimpson. Code is adapted from [diversity](#) and uses [sparseMatrix](#) in triplet format over the dense matrix. The code is much faster and memory efficient, while still being mathematically correct. This function is built into the class [omics](#) with method `alpha_diversity()` and inherited by other omics classes, such as [metagenomics](#) and [proteomics](#).

Usage

```
diversity(  
  x,  
  metric = c("shannon", "simpson", "invsimpson"),  
  normalize = TRUE,  
  base = exp(1)  
)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
metric	A character variable for metric; shannon, simpson or invsimpson.
normalize	A boolean variable for sample normalization by column sums.
base	Input for log to use natural logarithmic scale, log2, log10 or other (default: <code>exp(1)</code>).

Value

A numeric vector with type double.

See Also

[diversity](#)

Examples

```

n_row <- 1000
n_col <- 100
density <- 0.2
num_entries <- n_row * n_col
num_nonzero <- round(num_entries * density)

set.seed(123)
positions <- sample(num_entries, num_nonzero, replace=FALSE)
row_idx <- ((positions - 1) %% n_row) + 1
col_idx <- ((positions - 1) %% n_row) + 1

values <- runif(num_nonzero, min = 0, max = 1)
sparse_mat <- sparseMatrix(
  i = row_idx,
  j = col_idx,
  x = values,
  dims = c(n_row, n_col)
)

# Alpha diversity is computed on column level
## Transpose the sparseMatrix if required with t() from Matrix R package.
result <- OmicFlow::diversity(
  x = sparse_mat,
  metric = "shannon"
)

```

diversity_plot

Diversity plot

Description

Creates an Alpha diversity plot. This function is built into the class `omics` with method `alpha_diversity()`. It computes the pairwise wilcox test, paired or non-paired, given a data frame and adds useful labelling.

Usage

```

diversity_plot(
  data,
  values,
  col_name,
  group_by = NULL,
  palette,
  method,
  paired = FALSE,
  p.adjust.method = "fdr"
)

```

Arguments

<code>data</code>	A data.frame or data.table computed from diversity .
<code>values</code>	A column name of a continuous variable.
<code>col_name</code>	A column name of a categorical variable.
<code>group_by</code>	A column name to perform grouped statistical test (default: NULL).
<code>palette</code>	An object with names and hexcode or color names, see colormap .
<code>method</code>	A character variable indicating what method is used to compute the diversity.
<code>paired</code>	A boolean value to perform paired analysis in wilcox.test .
<code>p.adjust.method</code>	A character variable to specify the <code>p.adjust.method</code> to be used (Default: <code>fdr</code>).

Value

A [ggplot2](#) object to be further modified

Examples

```
library("ggplot2")

n_row <- 1000
n_col <- 100
density <- 0.2
num_entries <- n_row * n_col
num_nonzero <- round(num_entries * density)

set.seed(123)
positions <- sample(num_entries, num_nonzero, replace=FALSE)
row_idx <- ((positions - 1) %% n_row) + 1
col_idx <- ((positions - 1) %% n_row) + 1

values <- runif(num_nonzero, min = 0, max = 1)
sparse_mat <- Matrix::sparseMatrix(
  i = row_idx,
  j = col_idx,
  x = values,
  dims = c(n_row, n_col)
)

div <- OmicFlow::diversity(
  x = sparse_mat,
  metric = "shannon"
)

dt <- data.table::data.table(
  "shannon" = div,
  "treatment" = c(rep("healthy", n_col / 2), rep("tumor", n_col / 2)),
  "sex" = c(rep("male", n_col / 4), rep("female", n_col / 4))
)
```

```

colors <- OmicFlow::colormap(dt, "treatment")

# Comparing two groups
plt <- OmicFlow::diversity_plot(
  data = dt,
  values = "shannon",
  col_name = "treatment",
  palette = colors,
  method = "shannon",
  paired = FALSE,
  p.adjust.method = "fdr"
)

# Performing a test while stratifying the plot in two groups
plt <- OmicFlow::diversity_plot(
  data = dt,
  values = "shannon",
  col_name = "treatment",
  group_by = "sex",
  palette = colors,
  method = "shannon",
  paired = FALSE,
  p.adjust.method = "fdr"
)

```

euclidean

Compute Euclidean Dissimilarity from a from a Dense or Sparse Matrix.

Description

Calculates the Euclidean dissimilarity of a Matrix pairwise for each column.

Usage

```
euclidean(x, weighted = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The Euclidean dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. When weighted is set to FALSE, counts are replaced by presence/absence data.

Value

A column x column [dist](#) object.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

euclidean(taxa$countData)
```

 hill_taxa

Sparse implementation of Hill numbers

Description

Computes the hill numbers for q is 0, 1 or 2. Code is adapted from [hill_taxa](#) and uses [sparseMatrix](#) in triplet format over the dense matrix. The code is much faster and memory efficient, while still being mathematical correct.

Usage

```
hill_taxa(x, q = 0, normalize = TRUE, base = exp(1))
```

Arguments

<code>x</code>	A matrix , sparseMatrix or Matrix .
<code>q</code>	A wholenumber for 0, 1 or 2, default is 0.
<code>normalize</code>	A boolean variable for sample normalization by column sums.
<code>base</code>	Input for log to use natural logarithmic scale, log2, log10 or other.

Value

A numeric vector with type double.

See Also[hill_taxa](#)**Examples**

```

library("Matrix")

n_row <- 1000
n_col <- 100
density <- 0.2
num_entries <- n_row * n_col
num_nonzero <- round(num_entries * density)

set.seed(123)
positions <- sample(num_entries, num_nonzero, replace=FALSE)
row_idx <- ((positions - 1) %% n_row) + 1
col_idx <- ((positions - 1) %% n_row) + 1

values <- runif(num_nonzero, min = 0, max = 1)
sparse_mat <- sparseMatrix(
  i = row_idx,
  j = col_idx,
  x = values,
  dims = c(n_row, n_col)
)

result <- OmicFlow::hill_taxa(
  x = sparse_mat,
  q = 2
)

```

jaccard

*Compute Jaccard Dissimilarity from a Dense or Sparse Matrix.***Description**

Calculates the Jaccard dissimilarity of a Matrix pairwise for each column.

Usage

```
jaccard(x, weighted = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The weighted Jaccard dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = 1 - \frac{\sum_i^n \min(A_i, B_i)}{\sum_i^n \max(A_i, B_i)}$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. When `weighted` is set to `FALSE`, abundances are changed to 1 (classical Jaccard for binary data).

Value

A column x column `dist` object.

References

Jaccard, P. (1912) The distribution of the flora in the alpine zone. *New Phytologist*, 11(2), 37–50.
`library("OmicFlow")`

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")
```

```
taxa <- metagenomics$new( metaData = metadata_file, countData = counts_file, featureData = features_file, treeData = tree_file )
```

```
taxa$feature_subset(Kingdom == "Bacteria") taxa$scale(method = "tss")
```

```
jaccard(taxa$countData)
```

 jsd

Compute Jensen-Shannon Divergence from a Dense or Sparse Matrix.

Description

Calculates the Jensen-Shannon divergence of a Matrix pairwise for each column.

Usage

```
jsd(x, weighted = TRUE, threads = 1)
```

Arguments

<code>x</code>	A <code>matrix</code> , <code>sparseMatrix</code> or <code>Matrix</code> .
<code>weighted</code>	A boolean value, to use abundances (<code>weighted = TRUE</code>) or absence/presence (<code>weighted=FALSE</code>) (default: <code>TRUE</code>).
<code>threads</code>	A wholenumber, the number of threads to use in <code>setThreadOptions</code> (default: 1).

Details

The Jensen-Shannon divergence between two probability distributions A and B , each of length n , is defined as:

$$d(A, B) = \frac{1}{2}D_{KL}(A \parallel M) + \frac{1}{2}D_{KL}(B \parallel M)$$

where $M = \frac{1}{2}(A + B)$ is the mixture distribution, and D_{KL} is the Kullback-Leibler divergence. When `weighted` is set to `FALSE`, counts are replaced by presence/absence data.

Value

A column x column `dist` object.

References

Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1), 145-151.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

jsd(taxa$countData)
```

manhattan

Compute Manhattan Dissimilarity from a Dense or Sparse Matrix.

Description

Calculates the Manhattan dissimilarity of a Matrix pairwise for each column.

Usage

```
manhattan(x, weighted = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix .
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The Manhattan dissimilarity between two samples A and B , each of length n , is defined as:

$$d(A, B) = \sum_i^n |A_i - B_i|$$

where A_i and B_i are the abundances of the i -th feature in sample A and B , respectively. When weighted is set to FALSE, counts are replaced by presence/absence data.

Value

A column x column [dist](#) object.

References

Deza, M. M., & Deza, E. (2009). Encyclopedia of Distances. Springer Science & Business Media., 313.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

manhattan(taxa$countData)
```

matrix_to_dtable	<i>Converting a Matrix to data.table</i>
------------------	--

Description

Wrapper function that converts a sparseMatrix to data.table

Usage

```
matrix_to_dtable(x)
```

Arguments

x A [matrix](#), [sparseMatrix](#) or [Matrix](#).

Value

A [data.table](#) class.

metagenomics	<i>Sub-class metagenomics</i>
--------------	-------------------------------

Description

This is a sub-class that is compatible to data obtained from either 16S rRNA marker-gene sequencing or shot-gun metagenomics sequencing. It inherits all methods from the abstract class [omics](#) and only adapts the `initialize` function. It supports BIOM format data (v2.1.0 from <http://biom-format.org/>) in both HDF5 and JSON format, also pre-existing data structures can be used or text files. When omics data is very large, data loading becomes very expensive. It is therefore recommended to use the `reset()` method to reset your changes. Every omics class creates an internal memory efficient back-up of the data, the resetting of changes is an instant process.

Value

- data A long [data.table](#) table.
- volcano_plot A [ggplot](#) object.
- A A [data.table](#) table for (each) condition A.
- B A [data.table](#) table for (each) condition B.

Super class

`OmicFlow::omics` -> metagenomics

Active bindings

treeData A "phylo" class, see [as.phylo](#).

Methods

Public methods:

- `metagenomics$new()`
- `metagenomics$write_biom()`
- `metagenomics$foldchange()`
- `metagenomics$clone()`

Method `new()`: Initializes the metagenomics class object with `metagenomics$new()`

Usage:

```
metagenomics$new(
  countData = NULL,
  metaData = NULL,
  featureData = NULL,
  treeData = NULL,
  biomData = NULL,
  feature_names = c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species")
)
```

Arguments:

`countData` A path to an existing file or a dense/sparse [Matrix](#) format.

`metaData` A path to an existing file, [data.table](#) or `data.frame`.

`featureData` A path to an existing file, [data.table](#) or `data.frame`.

`treeData` A path to an existing newick file or class "phylo", see [read.tree](#).

`biomData` A path to an existing biom file, version 2.1.0 (<http://biom-format.org/>), see [h5read](#).

`feature_names` A character vector to name the feature names that fit the supplied `featureData`.

Returns: A new metagenomics object.

Method `write_biom()`: Creates a BIOM file in HDF5 format of the loaded items via `'new()'`, which is compatible to the python biom-format version 2.1, see <http://biom-format.org>.

Usage:

```
metagenomics$write_biom(filename)
```

Arguments:

`filename` A character variable of either the full path of filename of the biom file (e.g. `output.biom`)

Examples:

```
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")
```

```
taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
```

```

    treeData = tree_file
  )

  taxa$write_biom(filename = "output.biom")
  file.remove("output.biom")

```

Method `foldchange()`: Differential feature expression (DFE) Total Sample Sum (TSS) transformed values for both paired and non-paired test.

The function performs feature agglomeration, subsetting to remove NAs in `condition.group`, finding `samplepairs` and finally feature scaling prior to fold-change computation. Based on the transform method, fold-changes will be computed either via subtraction or division.

Usage:

```

metagenomics$foldchange(
  condition.group,
  condition_A,
  condition_B,
  group_by = NULL,
  feature_rank = "FEATURE_ID",
  feature_filter = NULL,
  paired = FALSE,
  normalize = FALSE,
  pvalue.threshold = 0.05,
  logfold.threshold = 0.06,
  abundance.threshold = 0
)

```

Arguments:

`condition.group` A character variable of an existing column name in `metaData`, wherein the conditions A and B are located.

`condition_A` A character value or vector of characters.

`condition_B` A character value or vector of characters.

`group_by` A character variable of an existing column in `metaData` to split the table in chunks prior to fold-change computation (default: `NULL`).

`feature_rank` A character or vector of characters in the `featureData` to aggregate via `feature_merge()` (default: `"FEATURE_ID"`).

`feature_filter` A character or vector of characters to remove features via regex pattern (default: `NULL`).

`paired` A boolean value, the `paired` is only applicable when a `SAMPLEPAIR_ID` column exists within the `metaData`. See `wilcox.test` and `samplepair_subset()`.

`normalize` A boolean value whether to normalize via Total Sample Sums (TSS) or not (default: `FALSE`).

`pvalue.threshold` A numeric value used as a p-value threshold to label and color significant features (default: 0.05).

`logfold.threshold` A numeric value used as a fold-change threshold to label and color significantly expressed features (default: 0.06).

`abundance.threshold` A numeric value used as an abundance threshold to size the scatter dots based on their mean abundance (default: 0.01).

Examples:

```
library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)

dfe <- obj$foldchange(feature_rank = "Genus",
  paired = FALSE,
  condition.group = "treatment",
  condition_A = c("healthy"),
  condition_B = c("tumor"))
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
metagenomics$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[omics](#)

[volcano_plot](#)

Examples

```
## -----
## Method `metagenomics$write_biom`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
```

```

    treeData = tree_file
  )

  taxa$write_biom(filename = "output.biom")
  file.remove("output.biom")

  ## -----
  ## Method `metagenomics$foldchange`
  ## -----

  library("ggplot2")
  library("OmicFlow")

  metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
  counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
  features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

  obj <- metagenomics$new(
    metaData = metadata_file,
    countData = counts_file,
    featureData = features_file
  )

  dfe <- obj$foldchange(feature_rank = "Genus",
                        paired = FALSE,
                        condition.group = "treatment",
                        condition_A = c("healthy"),
                        condition_B = c("tumor"))

```

omics

Abstract omics class

Description

This is the abstract class 'omics', contains a variety of methods that are inherited and applied in the omics classes: [metagenomics](#) and [proteomics](#).

Details

Every class is created with the [R6Class](#) method. Methods are either public or private, and only the public components are inherited by other omic classes. The omics class by default uses a [sparseMatrix](#) and [data.table](#) data structures for quick and efficient data manipulation and returns the object by reference, same as the R6 class. The method by reference is very efficient when dealing with big data.

Value

A list of components:

- div A [data.frame](#) from [diversity](#).
- stats A pairwise statistics from [pairwise_wilcox_test](#).
- plot A [ggplot](#) object.

A list of components:

- data A [data.table](#) of feature compositions.
- palette A [setNames](#) palette from [colormap](#).

A list of components:

- distmat A distance dissimilarity in [matrix](#) format.
- stats A statistical test as a [data.frame](#).
- pcs principal components as a [data.frame](#).
- scree_plot A [ggplot](#) object.
- anova_plot A [ggplot](#) object.
- scores_plot A [ggplot](#) object.

- data A long [data.table](#) table.
- volcano_plot A [ggplot](#) object.
- A A [data.table](#) table for (each) condition A
- B A [data.table](#) table for (each) condition B

Active bindings

metaData A [data.table](#) with SAMPLE_ID column.

featureData A [data.table](#) with FEATURE_ID column.

countData A dense or sparse [Matrix](#).

Methods**Public methods:**

- [omics\\$new\(\)](#)
- [omics\\$copy\(\)](#)
- [omics\\$validate\(\)](#)
- [omics\\$print\(\)](#)
- [omics\\$reset\(\)](#)
- [omics\\$removeNAs\(\)](#)
- [omics\\$feature_subset\(\)](#)
- [omics\\$sample_subset\(\)](#)
- [omics\\$samplepair_subset\(\)](#)
- [omics\\$feature_merge\(\)](#)

- `omics$scale()`
- `omics$rankstat()`
- `omics$alpha_diversity()`
- `omics$composition()`
- `omics$distance()`
- `omics$ordination()`
- `omics$foldchange()`
- `omics$autoFlow()`
- `omics$clone()`

Method `new()`: Wrapper function that is inherited and adapted for each omics class. The omics classes requires a metadata samplesheet, that is validated by the `metadata_schema.json`. It requires a column `SAMPLE_ID` and optionally a `SAMPLEPAIR_ID` can be supplied. The `SAMPLE_ID` will be used to link the `metaData` to the `countData`, and will act as the key during subsetting of other columns. To create a new object use `new()` method. Do notice that the abstract class only checks if the metadata is valid! The `countData` and `featureData` will not be checked, these are handled by the sub-classes. Using the omics class to load your data is not supported and still experimental.

Usage:

```
omics$new(countData = NULL, featureData = NULL, metaData = NULL)
```

Arguments:

`countData` A path to an existing file or a dense/sparse **Matrix** format.

`featureData` A path to an existing file, `data.table` or `data.frame`.

`metaData` A path to an existing file, `data.table` or `data.frame`.

Returns: A new omics object.

Method `copy()`: Create a copy of the object-class

This method is very similar to the existing `clone()` function, except it also resets the back-up of the OmicFlow data types that is invoked with `reset()`

Usage:

```
omics$copy(deep = FALSE)
```

Arguments:

`deep` A boolean value to create a shallow or deep copy.

Returns: A copy of omics object

Examples:

```
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
```

```
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
```

```
obj <- omics$new(
  metaData = metadata_file,
  countData = counts_file
)
```

```
# Perform a modification and copy
obj$scale()

cloned <- obj$copy(deep=TRUE)
cloned$scale(method = "clr")
cloned$reset() # resets to data after clone creation.
```

Method validate(): Validates an input metadata against the JSON schema. The metadata should look as follows and should not contain any empty spaces. For example; 'sample 1' is not allowed, whereas 'sample1' is allowed!

Acceptable column headers:

- SAMPLE_ID (required)
- SAMPLEPAIR_ID (optional)
- CONTRAST_ (optional), used for [autoFlow\(\)](#).
- VARIABLE_ (optional), not supported yet.

This function is used during the creation of a new object via [new\(\)](#) to validate the supplied metadata via a filepath or existing [data.table](#) or [data.frame](#).

Usage:

```
omics$validate()
```

Returns: None

Method print(): Displays parameters of the omics class via stdout.

Usage:

```
omics$print()
```

Returns: object in place

Examples:

```
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
```

```
obj <- omics$new(
  metaData = metadata_file,
  countData = counts_file
)
```

```
# method 1 to call print function
obj
```

```
# method 2 to call print function
obj$print()
```

Method reset(): Upon creation of a new omics object a small backup of the original data is created. Since modification of the object is done by reference and duplicates are not made,

it is possible to reset changes to the class. The methods from the abstract class `omics` also contains a private method to prevent any changes to the original object when using methods such as `ordination` `alpha_diversity` or `foldchange`.

Usage:

```
omics$reset()
```

Returns: object in place

Examples:

```
library(ggplot2)
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
```

```
taxa <- omics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)
```

```
# Performs modifications
taxa$scale(transform = log2)
```

```
# resets
taxa$reset()
```

```
# An inbuilt reset function prevents unwanted modification to the taxa object.
taxa$rankstat(feature_ranks = c("Kingdom", "Phylum", "Family", "Genus", "Species"))
```

Method `removeNAs()`: Remove NAs from `metaData` and updates the `countData`.

Usage:

```
omics$removeNAs(column)
```

Arguments:

`column` The column from where NAs should be removed, this can be either a wholenumbers or characters. Vectors are also supported.

Returns: object in place

Examples:

```
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
```

```
obj <- metagenomics$new(
```

```

    metaData = metadata_file,
    countData = counts_file,
    featureData = features_file,
  )

  obj$removeNAs(column = "treatment")

```

Method `feature_subset()`: Feature subset (based on `featureData`), automatically applies data synchronization.

Usage:

```
omics$feature_subset(...)
```

Arguments:

... Expressions that return a logical value, and are defined in terms of the variables in `featureData`. Only rows for which all conditions evaluate to TRUE are kept.

Returns: object in place

Examples:

```

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$feature_subset(Genus == "Pseudomonas")

```

Method `sample_subset()`: Sample subset (based on `metaData`), automatically applies synchronization.

Usage:

```
omics$sample_subset(...)
```

Arguments:

... Expressions that return a logical value, and are defined in terms of the variables in `metaData`. Only rows for which all conditions evaluate to TRUE are kept.

Returns: object in place

Examples:

```

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")

```

```

features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$sample_subset(treatment == "tumor")

```

Method `samplepair_subset()`: Samplepair subset (based on metaData), automatically applies synchronization.

Usage:

```
omics$samplepair_subset(num_unique_pairs = NULL)
```

Arguments:

`num_unique_pairs` An integer value to define the number of pairs to subset. The default is NULL, meaning the maximum number of unique pairs will be used to subset the data. Let's say you have three samples for each pair, then the `num_unique_pairs` will be set to 3.

Returns: object in place

Method `feature_merge()`: Agglomerates features by column, automatically applies synchronization.

Usage:

```
omics$feature_merge(feature_rank, feature_filter = NULL)
```

Arguments:

`feature_rank` A character value or vector of columns to aggregate from the featureData.

`feature_filter` A character value or vector of characters to remove features via regex pattern.

Returns: object in place

Examples:

```
library("OmicFlow")
```

```

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

```

```

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

```

```

obj$feature_merge(feature_rank = c("Kingdom", "Phylum"))
obj$feature_merge(feature_rank = "Genus", feature_filter = c("uncultured", "metagenome"))

```

Method `scale()`: Feature scaling on the `countData`. The `scale` function is able to apply transformations element-wise on the positive values, (optional: add pseudocounts) and perform normalisation or standardisation methods.

Usage:

```
omics$scale(
  method = "tss",
  transform = NULL,
  base = exp(1),
  pseudocount = NULL
)
```

Arguments:

`method` A character to choose a standardisation/normalisation method, options: `tss`, `clr`, `binary`, `hellinger`, `none` (default: `"tss"`).

`transform` A function to apply on the positive values of `countData`, skip standardisation/normalisation with `method = "none"` (default: `NULL`).

`base` Input for `log` to use natural logarithmic scale, `log2`, `log10` or other (default: `exp(1)`) in CLR.

`pseudocount` A numeric value to replace zero's (default: `NULL`).

Returns: object in place

Examples:

```
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
```

```
obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)
# standard relative abundance computation
obj$scale()

# CLR
obj$reset()
obj$scale(method = "clr")

# transform
obj$reset()
obj$scale(method = "none", transform = log2)
```

Method `rankstat()`: Rank statistics based on `featureData`

Usage:

```
omics$rankstat(feature_ranks, unique = FALSE)
```

Arguments:

`feature_ranks` A vector of characters or integers that match the `featureData`.

`unique` A boolean value to display only unique entries in `feature_ranks`.

Details: Counts the number of features identified for each column, for example in case of 16S metagenomics it would be the number of OTUs or ASVs on different taxonomy levels.

Returns: A `ggplot` object.

Examples:

```
library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

plt <- obj$rankstat(feature_ranks = c("Kingdom", "Phylum", "Family", "Genus", "Species"))
plt
```

Method `alpha_diversity()`: Alpha diversity based on [diversity](#)

Usage:

```
omics$alpha_diversity(
  col_name,
  metric = c("shannon", "invsimpson", "simpson"),
  Brewer.palID = "Set2",
  group_by = NULL,
  evenness = FALSE,
  paired = FALSE,
  p.adjust.method = "fdr"
)
```

Arguments:

`col_name` A character variable from the `metaData`.

`metric` An alpha diversity metric as input to [diversity](#).

`Brewer.palID` A character name for the palette set to be applied, see [brewer.pal](#) or [colormap](#).

`group_by` A column name to perform grouped statistical test in [diversity_plot](#) (default: `NULL`).

`evenness` A boolean wether to divide diversity by number of species, see [specnumber](#).

`paired` A boolean value to perform paired analysis in [wilcox.test](#) and samplepair subsetting via [samplepair_subset\(\)](#)

`p.adjust.method` A character variable to specify the `p.adjust.method` to be used, default is `'fdr'`.

Examples:

```

library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

plt <- obj$alpha_diversity(col_name = "treatment",
                          metric = "shannon")

```

Method `composition()`: Creates a table most abundant compositional features. Also assigns a color blind friendly palette for visualizations.

Usage:

```

omics$composition(
  feature_rank,
  feature_filter = NULL,
  col_name = NULL,
  feature_top = c(10, 15),
  Brewer.palID = "RdYlBu"
)

```

Arguments:

`feature_rank` A character variable in `featureData` to aggregate via `feature_merge()`.

`feature_filter` A character or vector of characters to removes features by regex pattern.

`col_name` Optional, a character or vector of characters to add to the final compositional data output.

`feature_top` A wholenumber of the top features to visualize, the max is 15, due to a limit of palettes.

`Brewer.palID` A character name for the palette set to be applied, see [brewer.pal](#) or [colormap](#).

Examples:

```

library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

```

```
)

result <- obj$composition(feature_rank = "Genus",
                          feature_filter = c("uncultured"),
                          feature_top = 10)

plt <- composition_plot(data = result$data,
                       palette = result$palette,
                       feature_rank = "Genus")
```

Method `distance()`: Compute a distance metric from `countData`

Usage:

```
omics$distance(
  metric,
  weighted = TRUE,
  threads = 1,
  normalized = TRUE,
  base = exp(1)
)
```

Arguments:

`metric` A dissimilarity metric to be applied on the `countData`, thus far supports 'bray', 'jaccard', 'cosine', 'manhattan', 'aitchison', 'euclidean', 'jsd' (jensen-shannon divergence), 'canberra' and 'unifrac' when a tree is provided via `treeData`, see [distance\(\)](#).

`weighted` A boolean value, to use abundances (`weighted = TRUE`) or absence/presence (`weighted=FALSE`) (default: `TRUE`).

`threads` A wholenumber, indicating the number of threads to use (Default: 1).

`normalized` A boolean value, whether to normalize weighted UniFrac distances to be between 0 and 1. Unweighted UniFrac is always normalized (default: `TRUE`).

`base` Input for [log](#) to use natural logarithmic scale, log2, log10 or other (default: `exp(1)`).

`pseudocount` A numeric value to replace zero's, used in [scale\(\)](#) (default: `1e-15`).

Returns: A column x column [dist](#) object.

Examples:

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)

obj$feature_subset(Kingdom == "Bacteria")
dist <- obj$distance(metric = "bray")
```


Method `foldchange()`: Differential feature expression (DFE) on log-transformed values for both paired and non-paired test.

The function performs feature agglomeration, subsetting to remove NAs in `condition.group` and finding samplepairs. It expects that the data is already log-transformed, this can be accomplished via `scale()`

Usage:

```
omics$foldchange(
  condition.group,
  condition_A,
  condition_B,
  group_by = NULL,
  feature_rank = "FEATURE_ID",
  feature_filter = NULL,
  paired = FALSE,
  pvalue.threshold = 0.05,
  logfold.threshold = 0.06,
  abundance.threshold = 0
)
```

Arguments:

`condition.group` A character variable of an existing column name in `metaData`, wherein the conditions A and B are located.

`condition_A` A character value or vector of characters.

`condition_B` A character value or vector of characters.

`group_by` A character variable of an existing column in `metaData` to split the table in chunks prior to fold-change computation (default: `NULL`). When disabled then column names will end with `_in_all`.

`feature_rank` A character or vector of characters in the `featureData` to aggregate via `feature_merge()` (default: `"FEATURE_ID"`).

`feature_filter` A character or vector of characters to remove features via regex pattern (default: `NULL`).

`paired` A boolean value, the paired is only applicable when a `SAMPLEPAIR_ID` column exists within the `metaData`. See `wilcox.test` and `samplepair_subset()`.

`pvalue.threshold` A numeric value used as a p-value threshold to label and color significant features (default: 0.05).

`logfold.threshold` A numeric value used as a fold-change threshold to label and color significantly expressed features (default: 0.06).

`abundance.threshold` A numeric value used as an abundance threshold to size the scatter dots based on their mean abundance (default: 0.01).

Examples:

```
library("ggplot2")
library("OmicFlow")
```

```
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
```

```

obj <- omics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)
obj$scale(method = "clr")

dfe <- obj$foldchange(feature_rank = "Genus",
  paired = FALSE,
  condition.group = "treatment",
  condition_A = c("healthy"),
  condition_B = c("tumor"))

```

Method `autoFlow()`: Automated Omics Analysis based on the `metaData`, see `validate()`. For now only works with headers that start with prefix `CONTRAST_`. If the data is from the class `omics` or proteomics than FDR adjusted p-values are computed for the volcano plots. Log-transformed values will lead to the skipping of `composition()` and `alpha_diversity()` methods.

Usage:

```

omics$autoFlow(
  feature_contrast = "FEATURE_ID",
  feature_filter = NULL,
  feature_ranks = NULL,
  distance_metrics = c("bray"),
  distmat = NULL,
  weighted = TRUE,
  pvalue.threshold = 0.05,
  logfold.threshold = 1,
  abundance.threshold = 0.01,
  perm = 999,
  threads = 1,
  report = TRUE,
  filename = paste0(getwd(), "/report.html")
)

```

Arguments:

`feature_contrast` A character vector of feature columns in the `featureData` to aggregate via `feature_merge()` (default: "FEATURE_ID").

`feature_filter` A character vector to filter unwanted features, (default: NULL).

`feature_ranks` A character vector as input to `rankstat()` (default: NULL).

`distance_metrics` A character vector specifying what (dis)similarity metrics to use (default: `c("bray")`) When you are working with log-transformed data it is advised to use the euclidean.

`distmat` A path to an existing file or a dense/sparse `Matrix` format (default: NULL).

`weighted` A boolean value, whether to compute weighted or unweighted dissimilarities (default: TRUE).

`pvalue.threshold` A numeric value, the p-value is used to include/exclude composition and foldchanges plots coming from alpha- and beta diversity analysis (default: 0.05).

`logfold.threshold` A numeric value used as a fold-change threshold to label and color significantly expressed features, see `foldchange()` (Default: 1).

`abundance.threshold` A numeric value used as an abundance threshold to size the scatter dots based on their mean abundance, see `foldchange()` (default: 0.01).

`perm` A wholenumber, number of permutations to compare against the null hypothesis of `adonis2` or `anosim` (default: 999).

`threads` Number of threads to use, only used in `distance()` when `distmat` is not supplied (default: 1).

`report` A boolean value to create a HTML markdown report (default: FALSE). If FALSE a nested list of the plots and data is returned.

`filename` A character to name the HTML report to be saved in the current working directory (default: `paste0(getwd(), "/report.html")`). The `getwd()` is required for `rmarkdown` to save it in the right path.

Returns: List of plots/data or rendered HTML report

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
omics$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Aitchison, J. (1986) The Statistical Analysis of Compositional Data. Chapman and Hall, London, 416 p.

See Also

[diversity_plot](#)

[composition_plot](#)

[bray](#), [canberra](#), [cosine](#), [jaccard](#), [jsd](#), [manhattan](#), [unifrac](#)

[ordination_plot](#), [plot_pairwise_stats](#), [pairwise_anosim](#), [pairwise_adonis](#)

[volcano_plot](#)

Examples

```
## -----
## Method `omics$copy`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
```

```

obj <- omics$new(
  metaData = metadata_file,
  countData = counts_file
)

# Perform a modification and copy
obj$scale()

cloned <- obj$copy(deep=TRUE)
cloned$scale(method = "clr")
cloned$reset() # resets to data after clone creation.

## -----
## Method `omics$print`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")

obj <- omics$new(
  metaData = metadata_file,
  countData = counts_file
)

# method 1 to call print function
obj

# method 2 to call print function
obj$print()

## -----
## Method `omics$reset`
## -----

library(ggplot2)
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

taxa <- omics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)

# Performs modifications

```

```

taxa$scale(transform = log2)

# resets
taxa$reset()

# An inbuilt reset function prevents unwanted modification to the taxa object.
taxa$rankstat(feature_ranks = c("Kingdom", "Phylum", "Family", "Genus", "Species"))

## -----
## Method `omics$removeNAs`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$removeNAs(column = "treatment")

## -----
## Method `omics$feature_subset`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$feature_subset(Genus == "Pseudomonas")

## -----
## Method `omics$sample_subset`
## -----

library("OmicFlow")

```

```

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$sample_subset(treatment == "tumor")

## -----
## Method `omics$feature_merge`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

obj$feature_merge(feature_rank = c("Kingdom", "Phylum"))
obj$feature_merge(feature_rank = "Genus", feature_filter = c("uncultured", "metagenome"))

## -----
## Method `omics$scale`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)
# standard relative abundance computation
obj$scale()

# CLR
obj$reset()

```

```

obj$scale(method = "clr")

# transform
obj$reset()
obj$scale(method = "none", transform = log2)

## -----
## Method `omics$rankstat`
## -----

library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

plt <- obj$rankstat(feature_ranks = c("Kingdom", "Phylum", "Family", "Genus", "Species"))
plt

## -----
## Method `omics$alpha_diversity`
## -----

library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

plt <- obj$alpha_diversity(col_name = "treatment",
                          metric = "shannon")

## -----
## Method `omics$composition`
## -----

library("ggplot2")

```

```

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
)

result <- obj$composition(feature_rank = "Genus",
  feature_filter = c("uncultured"),
  feature_top = 10)

plt <- composition_plot(data = result$data,
  palette = result$palette,
  feature_rank = "Genus")

## -----
## Method `omics$distance`
## -----

library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file
)

obj$feature_subset(Kingdom == "Bacteria")
dist <- obj$distance(metric = "bray")

## -----
## Method `omics$ordination`
## -----

library("ggplot2")
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

obj <- metagenomics$new(
  metaData = metadata_file,

```

```

    countData = counts_file,
    featureData = features_file,
  )

  pcoa_plots <- obj$ordination(metric = "bray",
                              method = "pcoa",
                              group_by = "treatment",
                              weighted = TRUE)

  pcoa_plots

  ## -----
  ## Method `omics$foldchange`
  ## -----

  library("ggplot2")
  library("OmicFlow")

  metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
  counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
  features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")

  obj <- omics$new(
    metaData = metadata_file,
    countData = counts_file,
    featureData = features_file
  )
  obj$scale(method = "clr")

  dfe <- obj$foldchange(feature_rank = "Genus",
                        paired = FALSE,
                        condition.group = "treatment",
                        condition_A = c("healthy"),
                        condition_B = c("tumor"))

```

ordination_plot

Ordination plot

Description

Creates an ordination plot pre-computed principal components from [wcmdscale](#). This function is built into the class [omics](#) with method `ordination()` and inherited by other omics classes, such as; [metagenomics](#) and [proteomics](#).

Usage

```

ordination_plot(
  data,
  col_name,

```

```
    pair,  
    dist_explained = NULL,  
    dist_metric = NULL  
  )
```

Arguments

data	A data.frame or data.table of Principal Components as columns and rows as loading scores.
col_name	A categorical variable to color the contrasts (e.g. "groups").
pair	A vector of character variables indicating what dimension names (e.g. PC1, NMDS2).
dist_explained	A vector of numeric values of the percentage dissimilarity explained for the dimension pairs, default is NULL.
dist_metric	A character variable indicating what metric is used (e.g. unfrac, bray-curtis), default is NULL.

Value

A [ggplot2](#) object to be further modified

Examples

```
library(ggplot2)  
  
# Mock principal component scores  
set.seed(123)  
mock_data <- data.frame(  
  SampleID = paste0("Sample", 1:10),  
  PC1 = rnorm(10, mean = 0, sd = 1),  
  PC2 = rnorm(10, mean = 0, sd = 1),  
  groups = rep(c("Group1", "Group2"), each = 5)  
)  
  
# Basic usage  
ordination_plot(  
  data = mock_data,  
  col_name = "groups",  
  pair = c("PC1", "PC2")  
)  
  
# Adding variance/dissimilarity explained.  
ordination_plot(  
  data = mock_data,  
  col_name = "groups",  
  pair = c("PC1", "PC2"),  
  dist_explained = c(45, 22),  
  dist_metric = "bray-curtis"  
)
```

pairwise_adonis *Pairwise adonis2 (PERMANOVA) computation*

Description

Computes pairwise [adonis2](#), given a distance matrix and a vector of labels. This function is built into the class [omics](#) with method `ordination()` and inherited by other omics classes, such as; [metagenomics](#) and [proteomics](#).

Usage

```
pairwise_adonis(
  x,
  groups,
  metadata = NULL,
  perm_design = NULL,
  p.adjust.method = "bonferroni",
  perm = 999
)
```

Arguments

<code>x</code>	A distance matrix in the form of dist . Obtained from a dissimilarity metric, in the case of similarity metric please use <code>1-dist</code>
<code>groups</code>	A character vector (column from a table) of labels.
<code>metadata</code>	A <code>data.table</code> or <code>data.frame</code> of extra metadata for <code>perm_design</code> (default: <code>NULL</code>).
<code>perm_design</code>	A function that takes a <code>data.frame</code> and constructs a permutation design with how (default: <code>NULL</code>).
<code>p.adjust.method</code>	P adjust method see p.adjust .
<code>perm</code>	Number of permutations to compare against the null hypothesis of <code>adonis2</code> (default: <code>perm=999</code>).

Value

A `data.frame` of

- pairs that are used
- Degrees of freedom (Df)
- Sums of Squares of `H_0`
- F.Model of `H_0`
- R2 of `H_0`
- p value of $F^p > F$
- p adjusted

See Also[adonis2](#)**Examples**

```
# Create random data
set.seed(42)
mock_data <- matrix(rnorm(15 * 10), nrow = 15, ncol = 10)

# Create euclidean dissimilarity matrix
mock_dist <- dist(mock_data, method = "euclidean")

# Define group labels, should be equal to number of columns and rows to dist
mock_groups <- rep(c("A", "B", "C"), each = 5)

# Compute pairwise adonis (PERMANOVA)
result <- pairwise_adonis(x = mock_dist,
                          groups = mock_groups,
                          p.adjust.method = "bonferroni",
                          perm = 99)
```

`pairwise_anosim`*Pairwise anosim (ANOSIM) computation*

Description

Computes pairwise [anosim](#), given a distance matrix and a vector of labels. This function is built into the class [omics](#) with method `ordination()` and inherited by other omics classes, such as [metagenomics](#) and [proteomics](#).

Usage

```
pairwise_anosim(
  x,
  groups,
  metadata = NULL,
  perm_design = NULL,
  p.adjust.method = "bonferroni",
  perm = 999
)
```

Arguments

<code>x</code>	A distance matrix in the form of dist . Obtained from a dissimilarity metric, in the case of similarity metric please use <code>1-dist</code>
<code>groups</code>	A vector (column from a table) of labels.
<code>metadata</code>	A <code>data.table</code> or <code>data.frame</code> of extra metadata for <code>perm_design</code> (default: <code>NULL</code>).

perm_design	A function that takes a data.frame and constructs a permutation design with how (default: NULL).
p.adjust.method	P adjust method see p.adjust
perm	Number of permutations to compare against the null hypothesis of anosim (default: perm=999).

Value

A [data.frame](#) of

- pairs that are used
- R2 of H_0
- p value of $F^p > F$
- p adjusted

See Also

[anosim](#)

Examples

```
# Create random data
set.seed(42)
mock_data <- matrix(rnorm(15 * 10), nrow = 15, ncol = 10)

# Create euclidean dissimilarity matrix
mock_dist <- dist(mock_data, method = "euclidean")

# Define group labels, should be equal to number of columns and rows to dist
mock_groups <- rep(c("A", "B", "C"), each = 5)

# Compute pairwise anosim
result <- pairwise_anosim(x = mock_dist,
                          groups = mock_groups,
                          p.adjust.method = "bonferroni",
                          perm = 99)
```

plot_pairwise_stats *Create pairwise stats plot*

Description

Creates a pairwise stats plot from [pairwise_adonis](#) or [pairwise_anosim](#) results. This function is built into the class [omics](#) with method `ordination()` and inherited by other omics classes, such as; [metagenomics](#) and [proteomics](#).

Usage

```
plot_pairwise_stats(  
  data,  
  stats_col,  
  group_col,  
  label_col,  
  y_axis_title = NULL,  
  plot_title = NULL  
)
```

Arguments

data	A data.frame or data.table .
stats_col	A column name of a continuous variable.
group_col	A column name of a categorical variable.
label_col	A column name of a categorical variable to label the bars.
y_axis_title	A character variable to name the Y - axis title (default: NULL).
plot_title	A character variable to name the plot title (default: NULL).

Value

A [ggplot2](#) object to be further modified

Examples

```
library("ggplot2")  
  
# Create random data  
set.seed(42)  
mock_data <- matrix(rnorm(15 * 10), nrow = 15, ncol = 10)  
  
# Create euclidean dissimilarity matrix  
mock_dist <- dist(mock_data, method = "euclidean")  
  
# Define group labels, should be equal to number of columns and rows to dist  
mock_groups <- rep(c("A", "B", "C"), each = 5)  
  
# Compute pairwise adonis  
adonis_res <- pairwise_adonis(x = mock_dist,  
                             groups = mock_groups,  
                             p.adjust.method = "bonferroni",  
                             perm = 99)  
  
# Compute pairwise anosim  
anosim_res <- pairwise_anosim(x = mock_dist,  
                             groups = mock_groups,  
                             p.adjust.method = "bonferroni",  
                             perm = 99)  
  
# Visualize PERMANOVA pairwise stats
```

```

plot_pairwise_stats(data = adonis_res,
                    group_col = "pairs",
                    stats_col = "F.Model",
                    label_col = "p.adj",
                    y_axis_title = "Pseudo F test statistic",
                    plot_title = "PERMANOVA")

# Visualize ANOSIM pairwise stats
plot_pairwise_stats(data = anosim_res,
                    group_col = "pairs",
                    stats_col = "anosimR",
                    label_col = "p.adj",
                    y_axis_title = "ANOSIM R statistic",
                    plot_title = "ANOSIM")

```

proteomics

Sub-class proteomics

Description

This is a sub-class that is compatible to preprocessed data obtained from <https://fragpipe.nesvilab.org/>. It inherits all methods from the abstract class `omics` and only adapts the `initialize` function. It supports pre-existing data structures or paths to text files. When omics data is very large, data loading becomes very expensive. It is therefore recommended to use the `reset()` method to reset your changes. Every omics class creates an internal memory efficient back-up of the data, the resetting of changes is an instant process.

Super class

`OmicFlow::omics` -> proteomics

Active bindings

`treeData` A "phylo" class, see [as.phylo](#).

Methods

Public methods:

- `proteomics$new()`
- `proteomics$clone()`

Method `new()`: Initializes the proteomics class object with `proteomics$new()`

Usage:

```

proteomics$new(
  countData = NULL,
  metaData = NULL,
  featureData = NULL,
  treeData = NULL
)

```

Arguments:

countData A path to an existing file or a dense/sparse [Matrix](#) format.

metaData A path to an existing file, [data.table](#) or data.frame.

featureData A path to an existing file, [data.table](#) or data.frame.

treeData A path to an existing newick file or class "phylo", see [read.tree](#).

Returns: A new proteomics object.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
proteomics$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[omics](#)

unifrac

Compute UniFrac Dissimilarity from a Dense or Sparse Matrix.

Description

Calculates the UniFrac dissimilarity between samples based on phylogenetic branch lengths and abundance or presence/absence data.

Usage

```
unifrac(x, tree, weighted = TRUE, normalized = TRUE, threads = 1)
```

Arguments

x	A matrix , sparseMatrix or Matrix of strictly positive counts or presence/absence data.
tree	A phylo class tree.
weighted	A boolean value, to use abundances (weighted = TRUE) or absence/presence (weighted=FALSE) (default: TRUE).
normalized	A boolean value, whether to normalize weighted UniFrac distances to be between 0 and 1 (default: TRUE). Unweighted UniFrac is always normalized.
threads	A wholenumber, the number of threads to use in setThreadOptions (default: 1).

Details

The UniFrac distance between two samples A and B , with phylogenetic tree edges $i = 1 \dots n$ of lengths L_i , is computed differently depending on the weighted and normalized flags. When `weighted = FALSE`, input counts are first converted to presence/absence data.

Weighted UniFrac (`normalized = FALSE` **and** `weighted = TRUE`): $d(A, B) = \frac{\sum_i^n L_i |A_i - B_i|}{\sum_i^n L_i (A_i + B_i)}$

Normalized Weighted UniFrac (`normalized = TRUE` **and** `weighted = TRUE`): $d(A, B) = \frac{\sum_i^n L_i |A_i - B_i|}{\sum_i^n L_i \max(A_i, B_i)}$

Unweighted UniFrac (`weighted = FALSE`, **unweighted is always normalized**): $d(A, B) = \frac{\sum_i^n L_i |A_i - B_i|}{\sum_i^n L_i \max(A_i, B_i)}$

Value

A column x column `dist` object.

References

Lozupone, C., & Knight, R. (2005). UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology*, 71(12), 8228–8235.

Examples

```
library("OmicFlow")

metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
features_file <- system.file("extdata", "features.tsv", package = "OmicFlow")
tree_file <- system.file("extdata", "tree.newick", package = "OmicFlow")

taxa <- metagenomics$new(
  metaData = metadata_file,
  countData = counts_file,
  featureData = features_file,
  treeData = tree_file
)

taxa$feature_subset(Kingdom == "Bacteria")
taxa$scale(method = "tss")

# Weighted UniFrac
unifrac(x = taxa$countData, tree = taxa$treeData, weighted=TRUE, normalized=FALSE)

# Weighted Normalized UniFrac
unifrac(x = taxa$countData, tree = taxa$treeData, weighted=TRUE, normalized=TRUE)

# Unweighted UniFrac
unifrac(x = taxa$countData, tree = taxa$treeData, weighted=FALSE)
```

volcano_plot	<i>Volcano plot</i>
--------------	---------------------

Description

Creates a Volcano plot from the output of `foldchange` method from class `omics`, it plots the fold-changes on the x-axis, log10 transformed p-values on the y-axis and adjusts the scatter size based on the percentage abundance of the features. This function is built into the class `omics` with method `DFE()` and inherited by other omics classes, such as; `metagenomics` and `proteomics`.

Usage

```
volcano_plot(  
  data,  
  logfold_col,  
  pvalue_col,  
  feature_rank,  
  abundance_col,  
  pvalue.threshold = 0.05,  
  logfold.threshold = 0.6,  
  abundance.threshold = 0.01,  
  label_A = "A",  
  label_B = "B"  
)
```

Arguments

<code>data</code>	A data.table .
<code>logfold_col</code>	A column name of a continuous variable.
<code>pvalue_col</code>	A column name of a continuous variable.
<code>feature_rank</code>	A character variable of the feature column.
<code>abundance_col</code>	A column name of a continuous variable.
<code>pvalue.threshold</code>	A P-value threshold (default: 0.05).
<code>logfold.threshold</code>	A $\text{Log}_2(A/B)$ Fold Change threshold (default: 0.6).
<code>abundance.threshold</code>	An abundance threshold (default: 0.01).
<code>label_A</code>	A character to describe condition A.
<code>label_B</code>	A character to describe condition B.

Value

A [ggplot2](#) object to be further modified.

Examples

```
library(data.table)
library(ggplot2)

# Create mock data frame
mock_volcano_data <- data.table(

  # Feature names (feature_rank)
  Feature = paste0("Gene", 1:20),

  # Log2 fold changes (X)
  log2FC = c(1.2, -1.5, 0.3, -0.7, 2.3,
            -2.0, 0.1, 0.5, -1.0, 1.8,
            -0.4, 0.7, -1.4, 1.5, 0.9,
            -2.1, 0.2, 1.0, -0.3, -1.8),

  # P-values (Y)
  pvalue = c(0.001, 0.02, 0.3, 0.04, 0.0005,
            0.01, 0.7, 0.5, 0.02, 0.0008,
            0.15, 0.06, 0.01, 0.005, 0.3,
            0.02, 0.8, 0.04, 0.12, 0.03),

  # Mean (relative) abundance for point sizing
  rel_abun = runif(20, 0.01, 0.1)
)

volcano_plot(
  data = mock_volcano_data,
  logfold_col = "log2FC",
  pvalue_col = "pvalue",
  abundance_col = "rel_abun",
  feature_rank = "Feature",
)
```

Index

adonis2, [33](#), [36](#), [44](#), [45](#)
anosim, [33](#), [36](#), [45](#), [46](#)
as.phylo, [18](#), [48](#)

bray, [2](#), [36](#)
brewer.pal, [5](#), [30](#), [31](#)

canberra, [3](#), [36](#)
colormap, [5](#), [6](#), [11](#), [23](#), [30](#), [31](#)
column_exists, [5](#)
composition_plot, [6](#), [36](#)
cosine, [8](#), [36](#)

data.frame, [5](#), [6](#), [11](#), [23](#), [25](#), [43](#), [44](#), [46](#), [47](#)
data.table, [5](#), [6](#), [11](#), [18](#), [19](#), [22–25](#), [43](#), [47](#),
[49](#), [51](#)
dist, [3](#), [4](#), [8](#), [13](#), [15–17](#), [32](#), [33](#), [44](#), [45](#), [50](#)
diversity, [9](#), [9](#), [11](#), [23](#), [30](#)
diversity_plot, [10](#), [30](#), [36](#)

euclidean, [12](#)

ggplot, [6](#), [18](#), [23](#), [30](#)
ggplot2, [6](#), [11](#), [43](#), [47](#), [51](#)

h5read, [19](#)
hill_taxa, [13](#), [13](#), [14](#)
how, [33](#), [44](#), [46](#)

jaccard, [14](#), [36](#)
jsd, [15](#), [36](#)

log, [9](#), [13](#), [29](#), [32](#)

manhattan, [16](#), [36](#)
Matrix, [2](#), [4](#), [8](#), [9](#), [12–15](#), [17–19](#), [23](#), [24](#), [33](#),
[35](#), [49](#)
matrix, [2](#), [4](#), [8](#), [9](#), [12–15](#), [17](#), [18](#), [23](#), [49](#)
matrix_to_dtable, [18](#)
metagenomics, [5](#), [9](#), [18](#), [22](#), [42](#), [44–46](#), [51](#)

OmicFlow::omics, [18](#), [48](#)

omics, [5](#), [6](#), [9](#), [10](#), [18](#), [21](#), [22](#), [26](#), [42](#), [44–46](#),
[48](#), [49](#), [51](#)
ordination_plot, [36](#), [42](#)

p.adjust, [44](#), [46](#)
pairwise_adonis, [33](#), [36](#), [44](#), [46](#)
pairwise_anosim, [33](#), [36](#), [45](#), [46](#)
pairwise_wilcox_test, [23](#)
plot_pairwise_stats, [36](#), [46](#)
proteomics, [5](#), [9](#), [22](#), [42](#), [44–46](#), [48](#), [51](#)

R6Class, [22](#)
read.tree, [19](#), [49](#)

setNames, [5](#), [23](#)
setThreadOptions, [2](#), [4](#), [8](#), [12](#), [14](#), [15](#), [17](#), [49](#)
sparseMatrix, [2](#), [4](#), [8](#), [9](#), [12–15](#), [17](#), [18](#), [22](#), [49](#)
specnumber, [30](#)

unifrac, [36](#), [49](#)

volcano_plot, [21](#), [36](#), [51](#)

wcmdscale, [33](#), [42](#)
wilcox.test, [11](#), [20](#), [30](#), [34](#)