Package 'OmicFlow'

October 11, 2026

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

Version 1.7.0

Date 2026-10-11

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://agusinac.github.io/OmicFlow/

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

Config/testthat/edition 3

Encoding UTF-8

Depends R (>= 4.3.0)

Imports ape, cli, data.table, ggrepel, ggplot2, grDevices, jsonvalidate, lifecycle, methods, Matrix, matrixTests, matrixStats, patchwork, R6, RColorBrewer, rhdf5, Rcpp (>= 0.12.6), RcppParallel (>= 4.3.20), stats, utils, vegan, yyjsonr

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

LinkingTo Rcpp, RcppParallel, RcppArmadillo

Config/roxygen2/version 8.1.0

Config/Needs/website rmarkdown

NeedsCompilation yes

Author Alem Gusinac [aut, cre] (ORCID: <https://orcid.org/0009-0006-1896-4176>), Thomas Ederveen [aut] (ORCID: <https://orcid.org/0000-0003-0068-1275>), Annemarie Boleij [aut, fnd] (ORCID: <https://orcid.org/0000-0003-4495-5880>)

Maintainer Alem Gusinac <alem.gusinac@gmail.com>

Repository CRAN

Date/Publication 2026-10-11 13:50:08 UTC

Contents

boxjitter_test . . . 2
bray . . . 5
canberra . . . 6
colormap . . . 7
column_exists . . . 8
composition_plot . . . 8
cosine . . . 10
diversity . . . 11
euclidean . . . 13
hill_taxa . . . 14
jaccard . . . 15
jsd . . . 16
manhattan . . . 17
matrix_to_dtable . . . 19
metagenomics . . . 19
omics . . . 24
ordination_plot . . . 49
pairwise_adonis . . . 51
pairwise_anosim . . . 52
plot_pairwise_stats . . . 54
proteomics . . . 55
unifrac . . . 59
volcano_plot . . . 61

Index **63**

`boxjitter_test` *Create a boxplot with jittered points and statistical tests*

Description

Visualizes group differences using a half-boxplot, half-jitter plot with pairwise statistical testing and p-value annotation. This function is built into the class omics with method `alpha_diversity()`.

Usage

```
boxjitter_test(
  data,
  values,
  col_name = lifecycle::deprecated(),
  groups,
  group_by = lifecycle::deprecated(),
  split_by = NULL,
  palette = NULL,
  method = lifecycle::deprecated(),
  test = "wilcox",
  paired = FALSE,
  p.adjust.method = "fdr",
  step.increase = 0.05
)
```

Arguments

`data`	A data.frame or data.table table.
`values`	A column name of a continuous variable in `data`.
`col_name`	**[Deprecated]** This argument has been renamed to `groups` for more clarity.
`groups`	A column name of a categorical variable in `data`.
`group_by`	**[Deprecated]** This argument has been renamed to `split_by` for more clarity.
`split_by`	A column name to split the groups into chunks based on the `data` for grouped statistical testing (default: `NULL`).
`palette`	An setNames class with names and values as hexcode or color names, see colormap (default: `NULL`).
`method`	**[Deprecated]** This argument no longer labels the y-labels, you can accomplish this by adding `+ labs(y = "label")` to the returned plot.
`test`	A character variable indicating what statistical test to apply (default: `"wilcox"`).
`paired`	A boolean value to perform paired analysis with row_wilcoxon_paired (default: `FALSE`).
`p.adjust.method`	A character variable to adjust the p-values, see p.adjust (default: `"fdr"`).
`step.increase`	A numeric value to create a gap between Y-positions (default": `0.05`).

Value

A ggplot2 object to be further modified

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 <- 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::boxjitter_test(
 data = dt,
 values = "shannon",
 groups = "treatment",
 palette = colors,
 test = "wilcox",
 paired = FALSE,
 p.adjust.method = "fdr"
)

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

bray *Compute Bray-Curtis Dissimilarity from a Dense or Sparse Matrix.*

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` *Compute Canberra Dissimilarity from a from a Dense or Sparse Matrix.*

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:

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

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` *Color map of a variable*

Description

Creates a color map of setNames class with names as unique group labels given a column name and values as hexcode colors. This function is built into the `composition` method from the abstract class omics and inherited by other omics classes, such as; metagenomics and proteomics.

Usage

```
colormap(
  data,
  col_name = lifecycle::deprecated(),
  groups,
  Brewer.palID = "Set2"
)
```

Arguments

`data`	A data.frame or data.table.
`col_name`	**[Deprecated]** This argument has been renamed to `groups` for more clarity.
`groups`	A column name of a categorical variable in `data`.
`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,
                   groups = "treatment")
```

`column_exists` *Checks if given column name exists in provide table*

Description

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

Usage

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

Arguments

`column`	A column name in `table`.
`table`	A data.table or data.frame.

Value

A boolean value.

`composition_plot` *Compositional plot*

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 setNames class with names and values as hexcode or color names, see colormap.
feature_rank	A column name containing the feature names in data.
title_name	A character to set the ggtitle of the ggplot, (default: NULL).
group_by	A character variable to aggregate the stacked bars by groups (default: NULL).

Value

A ggplot2 object to be further modified

Examples

```
# 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 disimilarity of a Marix 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 = "shannon", normalize = TRUE, base = exp(1))
```

Arguments

x	A matrix, sparseMatrix or Matrix.
metric	A character to specify what metric to use, options: "shannon", "simpson", "invsimpson" (default: "shannon").
	"shannon" Shannon Diversity Index, $H = -\sum_i p_i \log_{\text{base}}(p_i)$, where p_i is the abundance or relative abundance of feature i. When normalize = TRUE, counts are converted to within-sample relative abundances before calculation. The logarithm base is controlled by base; the natural logarithm is commonly used.
	"simpson" Simpson Diversity Index, $1 - \sum_i p_i^2$. Higher values indicate greater diversity, with values near zero indicating domination by one feature. When normalize = TRUE, p_i denotes within-sample relative abundance.
	"invsimpson" Inverse Simpson Index, $1 / \sum_i p_i^2$. When calculated from relative abundances, this is the effective number of equally abundant features that would yield the observed concentration. Values increase with richness and evenness.
normalize	A boolean variable to normalise the each value by their column sum (default: TRUE).
base	A numeric variable in log to use natural logarithmic scale, log2, log10 or other in "shannon" (default: exp(1)).

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 <- Matrix::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"
)
```

`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

`x`	A matrix, sparseMatrix or Matrix.
`q`	A wholenumber to specify what hill numbers to use (default: `0`)
`normalize`	A boolean variable to normalise the each value by their column sum (default: `TRUE`).
`base`	A numeric variable in log to use natural logarithmic scale, log2, log10 or other in `"shannon"` (default: `exp(1)`).

Value

A numeric vector with type double.

See Also

hill_taxa

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 <- Matrix::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 disimilarity 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

`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 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 *Converting a Matrix to data.table*

Description

Wrapper function that converts a sparseMatrix or `countData` field from omics classes to a `data.table`

Usage

```
matrix_to_dtable(x)
```

Arguments

x	A matrix, sparseMatrix or Matrix.

Value

A data.table class.

metagenomics *Sub-class metagenomics*

Description

OmicFlow is constructed around the main superclass omics that in turn is inherited by metagenomics and proteomics, that offer additional fields or methods. The superclass omics contains both public and private methods, only the public componenents are documented, whereas the private methods can be accessed via `metagenomics$private_methods`, which is not supported.

The omics class components use Matrix and data.table in the background for fast loading and data wrangling, these data structures return the data by reference, so does the R6 class, therefore classes in OmicFlow do not create copies of the object unless called via `$copy()`.

All classes needs to be first initialised via the `$new()` method and require the `metaData` and `countData` (or `biomData` in case of `metagenomics`) field components. Alternatively, you can also initialise with only the `metaData` and later add the other fields via the active binding, but this will not create a back-up and is not advised! For more hands-on examples see Getting Started with OmicFlow.

Additional arguments in `metagenomics$new()`: The `metagenomics` class also offers the loading of a phylogenetic tree as a `phylo` class (see as.phylo) that can be supplied to `treeData`. When a `treeData` is supplied than every class field is arranged according to the tree tip labels.

The `biomData` argument expects a BIOM v2.1.0. in either HDF5 or JSON format. The `biomData` must be supplied via a filepath and required contents are checked upfront, such as the existence of `"/observation/matrix/data","/observation/matrix/indptr","/observation/matrix/indices","/observatio`

.

Input requirements

Every class in OmicFlow requires at least a `metaData` and `countData` field to be specified. The `metaData` must contain a `SAMPLE_ID` and optionally a `SAMPLEPAIR_ID`, see section "Metadata validation" below.

The `countData` can be both a matrix or data frame. If the original contains rownames, then these will become the `FEATURE_ID` in `featureData` (automatically created), otherwise a random feature identifier is assigned. If the `countData` is a filetype or a data frame with both matching sample identifiers in `SAMPLE_ID` and other additional columns, then these will be automatically split, columns matching the `SAMPLE_ID` will be placed in the `countData`, any other columns will be placed in front of the `FEATURE_ID` column in `featureData`.

Metadata validation

Every class in OmicFlow begins by validating the sample metadata. This validation checks that sample identifiers match the sample IDs present in the abundance tables.

The validation rules are defined in JSON format in the omics class. Metadata can be supplied as either:

- A CSV or TSV file.
- A `data.table`.

In both cases, metadata must include a header row and follow a one-row-per-sample structure. For file inputs, the header must be on the first line. Columns not described below are permitted and are ignored during metadata validation.

Minimum requirement: The metadata must contain a `SAMPLE_ID` column.

- Each row must have a unique, non-empty sample identifier.
- Sample identifiers cannot contain spaces; use underscores (_) or dashes (-) instead.

Example metadata:

SAMPLE_ID	SAMPLEPAIR_ID	CONTRAST_Treatment	VARIABLE_Age
S1	P1	Drug	42
S2	P1	Placebo	36
S3	P2	Drug	51

Column types and naming rules:

Required column

Column	Type	Rules
SAMPLE_ID	string	Unique, non-empty, no spaces; one identifier per sample row

Optional standard columns

Column	Type	Rules
SAMPLEPAIR_ID	string	Optional; no spaces. Used for samples that are paired or originate from the same individual or subj

Some functions require a `SAMEPLEPAIR_ID` in the `metaData` when `paired = TRUE` is used. The user can add or remove the `SAMPLEPAIR_ID` at anytime to the `metaData` via active binding.

Pattern-based columns

Additional variables can be defined with special column-name prefixes:

- `CONTRAST_...`: Grouping or category labels for differential comparisons. For example, `CONTRAST_Treatment` may contain `Drug` and `Placebo`.
- `VARIABLE_...`: Numeric or string variables for statistical analysis. For example, `VARIABLE_Age` may contain values such as `42` and `51`.

Pattern-based columns are only used in `$autoFlow()`. Currently, only columns with the `CONTRAST_` prefix are supported.

Super class

`omics` -> `metagenomics`

Active bindings

`treeData` A "phylo" class, see as.phylo.

Methods

Public methods:

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

`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, Matrix, data.table or data.frame (default: `NULL`).

`metaData` A path to an existing file, data.table or data.frame (default: `NULL`).

`featureData` A path to an existing file, data.table or data.frame (default: `NULL`).

`treeData` A path to an existing newick file or class "phylo", see read.tree (default: `NULL`).

`biomData` A path to an existing biom file, version 2.1.0 (see h5read) or JSON format (see validate_json_file) (default: `NULL`).

`feature_names` A character vector to name the feature names that fit the supplied `featureData` (default: `c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species")`).

Returns: A new metagenomics object.

Examples:

```
library("OmicFlow")

## Method 1: Load from filepath
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,  # optional
 treeData = tree_file          # optional
)

## Method 2: Load from a biom file
biom_file <- system.file("extdata", "biomv2.biom", package = "OmicFlow")

taxa <- metagenomics$new(
 metaData = metadata_file,
 biomData = biom_file
)
```

`metagenomics$write_biom()`**:** Creates a BIOM file in HDF5 format, which is compatible to the python biom-format v2.1.0. It only saves the `featureData` and `countData` to the biom format, no `treeData` is supported thus far.

Usage:

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

Arguments:

`filename` A character variable of a non-existing file path (e.g. `"output.biom"` or `"my/path/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")
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `metagenomics$new()`
## ------------------------------------------------

library("OmicFlow")

## Method 1: Load from filepath
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,  # optional
 treeData = tree_file          # optional
)

## Method 2: Load from a biom file
biom_file <- system.file("extdata", "biomv2.biom", package = "OmicFlow")

taxa <- metagenomics$new(
 metaData = metadata_file,
 biomData = biom_file
)

## ------------------------------------------------
## 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")
```

`omics` *Superclass omics*

Description

OmicFlow is constructed around the main superclass `omics` that in turn is inherited by metagenomics and proteomics, that offer additional fields or methods. The superclass `omics` contains both public and private methods, only the public componenents are documented, whereas the private methods can be accessed via `omics$private_methods`, which is not supported.

The `omics` class components use Matrix and data.table in the background for fast loading and data wrangling, these data structures return the data by reference, so does the R6 class, therefore classes in OmicFlow do not create copies of the object unless called via `$copy()`.

All classes needs to be first initialised via the `$new()` method and require the `metaData` and `countData` field components. Alternatively, you can also initialise with only the `metaData` and later add the other fields via the active binding, but this will not create a back-up and is not advised! For more hands-on examples see Getting Started with OmicFlow.

Input requirements

Every class in OmicFlow requires at least a `metaData` and `countData` field to be specified. The `metaData` must contain a `SAMPLE_ID` and optionally a `SAMPLEPAIR_ID`, see section "Metadata validation" below.

The `countData` can be both a matrix or data frame. If the original contains rownames, then these will become the `FEATURE_ID` in `featureData` (automatically created), otherwise a random feature identifier is assigned. If the `countData` is a filetype or a data frame with both matching sample identifiers in `SAMPLE_ID` and other additional columns, then these will be automatically split, columns matching the `SAMPLE_ID` will be placed in the `countData`, any other columns will be placed in front of the `FEATURE_ID` column in `featureData`.

Metadata validation

Every class in OmicFlow begins by validating the sample metadata. This validation checks that sample identifiers match the sample IDs present in the abundance tables.

The validation rules are defined in JSON format in the omics class. Metadata can be supplied as either:

- A CSV or TSV file.
- A `data.table`.

In both cases, metadata must include a header row and follow a one-row-per-sample structure. For file inputs, the header must be on the first line. Columns not described below are permitted and are ignored during metadata validation.

Minimum requirement: The metadata must contain a `SAMPLE_ID` column.

- Each row must have a unique, non-empty sample identifier.
- Sample identifiers cannot contain spaces; use underscores (_) or dashes (-) instead.

Example metadata:

SAMPLE_ID	SAMPLEPAIR_ID	CONTRAST_Treatment	VARIABLE_Age
S1	P1	Drug	42
S2	P1	Placebo	36
S3	P2	Drug	51

Column types and naming rules:

Required column

Column	Type	Rules
SAMPLE_ID	string	Unique, non-empty, no spaces; one identifier per sample row

Optional standard columns

Column	Type	Rules
SAMPLEPAIR_ID	string	Optional; no spaces. Used for samples that are paired or originate from the same individual or subj

Some functions require a `SAMEPLEPAIR_ID` in the `metaData` when `paired = TRUE` is used. The user can add or remove the `SAMPLEPAIR_ID` at anytime to the `metaData` via active binding.

Pattern-based columns

Additional variables can be defined with special column-name prefixes:

- `CONTRAST_...`: Grouping or category labels for differential comparisons. For example, `CONTRAST_Treatment` may contain `Drug` and `Placebo`.
- `VARIABLE_...`: Numeric or string variables for statistical analysis. For example, `VARIABLE_Age` may contain values such as `42` and `51`.

Pattern-based columns are only used in `$autoFlow()`. Currently, only columns with the `CONTRAST_` prefix are supported.

Active bindings

`metaData` A data.table with `SAMPLE_ID` column.

`featureData` A data.table with `FEATURE_ID` column.

`countData` A sparseMatrix with rownames matching `FEATURE_ID` of `featureData` and colnames matching `SAMPLE_ID` of `countData`.

Methods

Public methods:

- omics$new()
- omics$copy()
- 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()

omics$new():

Usage:

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

Arguments:

metaData A path to an existing file, data.table or data.frame (default: NULL).

countData A path to an existing file, Matrix, data.table or data.frame (default: NULL).

featureData A path to an existing file, data.table or data.frame (default: NULL).

Returns: A new omics object.

Examples:

```
library("OmicFlow")

## Method 1: load from filepath
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 2: Load from data.frame or matrix
n_cols <- 5
n_rows <- 100
```

```
n_vals <- n_cols * n_rows
metadata <- data.frame("SAMPLE_ID" = paste0("Sample_", 1:n_cols))
features <- data.frame("FEATURE_ID" = paste0("protein_", 1:n_rows))
counts <- Matrix::Matrix(
 1:n_vals, nrow = n_rows, ncol = n_cols,
 dimnames = list(features$FEATURE_ID, metadata$SAMPLE_ID)
)

obj <- omics$new(
 metaData = metadata,
 featureData = features, # optional
 countData = counts
)

## Method 3: you have features and counts in a single data.frame
metadata <- data.frame(SAMPLE_ID = c("S1", "S2", "S3"))
counts <- data.frame(
 S1 = c(2,3,0),
 S2 = c(2,"",1),
 S3 = c(2,1,NA),
 proteins = c("ZEB1", "ZEB2", "MAPK")
)

obj <- omics$new(
 metaData = metadata,
 countData = counts
)
```

`omics$copy()`: This method is very similar to the existing `$clone()` function, except it also creates a new save of the back-up of each class that is created when `$new()` is called and can be reverted to via `$reset()`

Usage:

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

Arguments:

`deep` A boolean value to create a shallow or deep copy (default: `FALSE`).

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.
```

`omics$print()`**:** Displays field parameters of the class via stdout.

Usage:

```
omics$print()
```

Returns: A cli formatted message

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()
```

`omics$reset()`**:** Upon creation of a new class object (i.e. `omics`, `metagenomics` or `proteomics`) a backup of the original data is created. Since modification of the object is done by reference and duplicates are not made, it is possible to revert back to the unmodified data via `$reset()`. The methods from the super 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: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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"))
```

`omics$removeNAs()`: Subset `metaData` via the complete.cases given a single or vector of column names, then synchronize the object.

Usage:

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

Arguments:

`column` A character vector of a single or multiple column names to remove NAs.

Returns: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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")
```

`omics$feature_subset()`: Subset features using a logical expression evaluated in `featureData`, then synchronize the object.

Usage:

```
omics$feature_subset(expr)
```

Arguments:

`expr` A logical expression evaluated in the context of `featureData`. Rows for which the expression evaluates to `TRUE` are retained.

Returns: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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")
```

`omics$sample_subset()`**:** Subset samples using a logical expression evaluated in `metaData`, then synchronize the object.

Usage:

```
omics$sample_subset(expr)
```

Arguments:

`expr` A logical expression evaluated in the context of `metaData`. Rows for which the expression evaluates to `TRUE` are retained.

Returns: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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")
```

`omics$samplepair_subset()`**:** Subset samplepairs by choosing the number of unique pairs in `metaData`, then synchronize the object.

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: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

`omics$feature_merge()`**:** Agglomerates features by column, then synchronize the object.

Usage:

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

Arguments:

`feature_rank` A character (vector) of columns to aggregate from the `featureData`.

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

Returns: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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"))
```

`omics$scale()`**:** Feature scaling on the `countData`. The `$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"`).

`"tss"` Total-sum scaling. Each column is divided by their column sum, converting absolute abundance to relative abundances in range from 0 to 1.

`"clr"` Centered log-ratio-like transformation. Non-zero entries are log-transformed using `log` and `base`, then each feature row is centered by subtracting by their respective geometric mean. Input counts must be strictly positive for his transformation. If `pseudocount`

is used, then output is equal to `vegan::decostand(method = "clr")`, otherwise the current `"clr"` approximates the `vegan::decostand(method = "rclr")`, the main difference is that vegan replaces all zero's by NAs and then excludes them when the arithmetic mean is taken, whereas zero's in `"clr"` are only not used for log-transformation but preserved when taking the arithmetic mean of each feature row.

`"binary"` Presence-absence transformation. Every stored non-zero count is replaced by one, while implicit zeros in the sparse matrix remain zero.

`"hellinger"` Hellinger transformation. Counts are first total-sum scaled with `"tss"` to relative abundances, after which the square root of each non-zero value is taken.

`"none"` No transformation. Typically applied when `transform` is used.

`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: The object is modified in place and returned (invisibly). If a copy is required, please use `$copy()` beforehand with `obj2 <- obj$copy()`.

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)
```

`omics$rankstat()`**:** Rank statistics based on `featureData`. Counts the number of (unique) features identified for each column of interest from the `featureData` (excluding the `FEATURE_ID`).

Usage:

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

Arguments:

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

`unique` A boolean value to display only unique entries in `feature_ranks` (default: `FALSE`).

Returns: A ggplot 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,
)

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

`omics$alpha_diversity()`**:** Alpha diversity computation, statistical testing and visualisation. It computes the alpha diversity via diversity, performs a pairwise wilcoxon statistical test given `groups` via the boxjitter_test.

Usage:

```
omics$alpha_diversity(
  col_name = lifecycle::deprecated(),
  groups,
  metric = "shannon",
  Brewer.palID = "Set2",
  group_by = lifecycle::deprecated(),
  split_by = NULL,
  evenness = FALSE,
  paired = FALSE,
  p.adjust.method = "fdr"
)
```

Arguments:

`col_name` **[Deprecated]** This argument has been renamed to `groups` for more clarity.

`groups` A column name in `metaData` to define the contrast of interest for statistical testing and visualisation.

`metric` An alpha diversity metric as input to diversity, options: `"shannon"`, `"simpson"`, `"invsimpson"` (default: `"shannon"`).

`Brewer.palID` A character name for the palette set to be applied, see brewer.pal or colormap (default: `"Set2"`).

`group_by` **[Deprecated]** This argument has been renamed to `split_by` for more clarity

`split_by` A column name to split the groups into chunks for grouped statistical test in boxjitter_test (default: `NULL`).

`evenness` A boolean wether to divide diversity by number of species, see specnumber (default: `FALSE`).

`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: `"fdr"`).

Returns: A list of components:

data A data.frame from diversity.

stats A pairwise statistics from pairwise_wilcox_test.

plot A ggplot 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,
)

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

`omics$composition()`**:** Creates a long `data.table` of `SAMPLE_IDs`, the top 10-15 most abundant bacteria based on the `feature_rank` with any additional columns specified via `add_cols`. The `$composition()` method works with aggregated values by running first `$feature_merge()`, then sorting for the most abundant bacteria based on the row sums. The output of `$composition()` can be used as input to composition_plot.

Usage:

```
omics$composition(
  feature_rank,
  feature_filter = NULL,
  col_name = lifecycle::deprecated(),
  add_cols = NULL,
  feature_top = 10,
  Brewer.palID = "RdYlBu"
)
```

Arguments:

`feature_rank` A column name in `featureData` to aggregate via `$feature_merge()`.

`feature_filter` A character or vector of characters to removes features by regex substring pattern matching via `$feature_merge()` (default: `NULL`).

`col_name` **[Deprecated]** This argument has been renamed to `add_cols` for more clarity.

`add_cols` Optional, a column name or vector of multiple column names from `metaData` to add to the `data` list name (default": `NULL`).

`feature_top` A wholenumber of the top features to visualize, the max is 15, due to a limit of palettes (default: `10`).

`Brewer.palID` A character name for the palette set to be applied, see brewer.pal or colormap (default: `"RdYlBu"`).

Returns: A list of components:

data A data.table of feature compositions.

palette A setNames palette from colormap matching the top features.

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,
)

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")
```

`omics$distance()`**:** Compute a distance metric from `countData`

Usage:

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

Arguments:

`metric` A dissimilarity metric to be applied on the `countData`, options:

- `"bray"` bray
- `"jaccard"` jaccard
- `"cosine"` cosine
- `"manhattan"` manhattan
- `"euclidean"` euclidean
- `"jsd"` jsd

"canberra" canberra

"unifrac" unifrac

"aitchison" The "aitchison" metric uses the $scale(method = "clr") without any pseudocounts, see $scale(), followed by euclidean dissimilarity computation.
Ref: Aitchison, J. (1986) The Statistical Analysis of Compositional Data. Chapman and Hall, London, 416 p.

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).

normalize 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)).

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")
```

omics$ordination(): The $ordination() method either computes a distance matrix by specification of the metric or takes a custom distance matrix via distmat. It then computes either the pairwise PERMANOVA test on the defined contrast via group_by via pairwise_adonis or the pairwise ANOSIM test via pairwise_anosim.

Which pairwise test to compute is decided via the method argument, where "pcoa" results in the pairwise_adonis and "nmds" to pairwise_anosim.

It also visualises these results as a scree-plot (only when "pcoa" is used), scores-plot (see ordination_plot) and an anova-plot (see plot_pairwise_stats). These plots, alongisde the distance matrix, principal components are returned as a list.

Usage:

```
omics$ordination(
  metric = "bray",
  method = "pcoa",
  group_by,
  distmat = NULL,
  weighted = TRUE,
  normalize = TRUE,
```

```
  threads = 1,
  perm_design = NULL,
  perm = 999
)
```

Arguments:

`metric` A dissimilarity or similarity metric to be applied on the `countData`, see `$distance()` (default: `"bray"`).

`method` Ordination method, options: `"pcoa"` or `"nmds"`, see wcmdscale (default: `"pcoa"`).

`"pcoa"` When `"pcoa"` (Principal Coordinate Analysis) is chosen, a pairwise PERMANOVA test via pairwise_adonis is conducted on the contrast specified by `group_by` with n number of permutations, `perm`.

`"nmds"` When `"nmds"` (Non-Metric Multidimensional Scaling) is chosen, a pairwise ANOSIM test via pairwise_anosim is conducted on the contrast specified by `group_by` with n number of permutations, `perm`. This method returns 2 dimensions by default, and a stress score in the `pcs` list name, that acts as the confidence, see metaMDS. Since only two dimensions are returned, there is no `scree_plot` output.

`group_by` A character variable in `metaData` to be used for the pairwise_adonis or pairwise_anosim statistical test.

`distmat` A custom distance matrix as either dist or Matrix (default: `NULL`).

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

`normalize` A boolean value, wether to normalize weighted UniFrac distances to be between 0 and 1, see unifrac (default: `TRUE`).

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

`perm_design` A function that takes `metaData` and constructs a permutation design with how, see example (default: `NULL`).

`perm` A whole number to define the number of permutations in adonis2 and anosim (default: `999`).

Returns: 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.

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,
)

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

`omics$foldchange()`: Differential feature expression for both paired and non-paired data.

The function performs feature agglomeration via `$feature_merge()` (which can be disabled via `feature_merge=FALSE`), subsetting to remove NAs in `condition.group` via `$removeNAs()` and finding samplepairs when `paired = TRUE` is supplied via `$subset_samplepairs()`.

The fold-changes can be computed differently based on the `method` and `aggregate_method` options. Transformations of the data should be done beforehand via `$scale()`. Finally, homogeneity of variance is computed based on the selected `aggregate_method` option. If `aggregate_method = "mean"` then the row_levene is applied, and if `aggregate_method = "median"` is used then the row_brownforsythe is applied. Any filtering of the results is left to the end-user, volcano plots are automatically added to the list name `volcano_plot` via volcano_plot.

Usage:

```
omics$foldchange(
  condition.group,
  condition_A,
  condition_B,
  method = "identity",
  aggregate_method = "median",
  group_by = lifecycle::deprecated(),
  split_by = NULL,
  feature_merge = FALSE,
  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 of an existing column name in `metaData`, wherein the conditions A and B are located.

`condition_A` A character (vector) in `condition.group`.

`condition_B` A character (vector) in `condition.group`.

`method` A character to choose the method of fold-change computation (default: `"identity"`).

- `"identity"` Computes fold-change via `log2(condition_A) - log2(condition_B)`, and will handle zero's to prevent `Inf` values. Proportional data is also supported and will be automatically detected.
- `"log"` Computes fold-change via `condition_A - condition_B`.

How to aggregate the fold-change values for `condition_A` and `condition_B` can be controlled with the `aggregate_method` argument.

`aggregate_method` A function to aggregate the fold-change values from multiple samples (default: `"median"`). options:

- `"mean"` It takes the arithmetic mean for each feature, separately for `condition_A` and `condition_B`.
- `"median"` It takes the median for each feature, separately for `condition_A` and `condition_B`.
- `"geomean"` It computes the geometric mean of the positive counts, similar to the estimateSizeFactors, when `type = "poscount"`. The only difference, is that the code is adapted to handle a sparseMatrix.
- `"none"` No aggreation is applied, the whole matrix is used as it is.

`group_by` **[Deprecated]** This argument has been renamed to `split_by` for more clarity.

`split_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_merge` A boolean value wether to call `$feature_merge()` (default: `FALSE`).

`feature_rank` A column in the `featureData` to use as the feature scope (default: `"FEATURE_ID"`).

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

`paired` A boolean value, the paired is only applicable when a `SAMPLEPAIR_ID` column exists within the `metaData` (default: `FALSE`), see `$samplepair_subset()`.

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

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

`abundance.threshold` A numeric value used as an abundance threshold to size the scatter dots based on their mean abundance (default: `0`), see volcano_plot.

Returns: A list of components:

`split_by` **subsets** A matrix subset from each `split_by` separated by `condition_A` and `condition_B`.

data A data.table main output of fold-changes between conditions, contains abundance, fold-change, and homogeneity tests.

volcano_plot A list of ggplot plots for each contrast of `A vs B`, number of plots depend on `split_by` and `condition_A` input.

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

```
## Method 1: using non-log transformation
obj$scale(method = "tss")

dfe <- obj$foldchange(
 feature_rank = "Genus",
 feature_merge = TRUE,
 method = "identity",
 aggregate_method = "median",
 condition.group = "treatment",
 condition_A = "tumor",
 condition_B = "healthy"
)

## Method 2: using log transformation
obj$reset()
obj$scale(method = "clr")

dfe <- obj$foldchange(
 feature_rank = "Genus",
 feature_merge = TRUE,
 method = "log",
 aggregate_method = "mean",
 condition.group = "treatment",
 condition_A = "tumor",
 condition_B = "healthy"
)
```

`omics$autoFlow()`**:** Automated Omics Analysis only works if there are column names in the `metaData` with the prefix `"CONTRAST_"`, which is detected via sub-string pattern matching. The transformation of the data is left to the user via the `$scale()` method. The `$autoFlow()` incorporates the following functions:

`$rankstat()` This function is only applied when `feature_ranks` is specified, useful in metagenomics, see `$rankstat()` for more information.

`$alpha_diversity()` This function only works with the `metagenomics` class, uses `"shannon"` metric, and works for both paired and non-paired data, applied on each `"CONTRAST_"` column, see `$alpha_diversity()` for more information.

`$ordination()` This function is applied on all classes, multiple `distance_metrics` can be supplied, and only the Principal Component method (`$ordination(method = "pcoa")`) is used. Additional arguments can be controlled with `distance_metrics`, `distmat`, `weighted`, `perm`, `threads`, see `$ordination()` for more information. **NOTE**: `perm_design` is not supported in `$autoFlow()`.

`$composition()` This function is applied on the significant contrasts that are obtained from either `$alpha_diversity()` or `$ordination()`. Multiple `feature_contrasts` can be supplied, by default it only uses the `feature_contrasts = "FEATURE_ID"`. The composition plots are created by default via the `composition_plot(group_by)` with `feature_top = 15`. Additional arguments can be controlled with `feature_filter`, see `$composition()` for more information.

`$foldchange()` This function is applied on the significant contrasts that are obtained from either

`$alpha_diversity()` or `$ordination()`. Multiple `feature_contrasts` can be supplied, by default it only uses the `feature_contrasts = "FEATURE_ID"`. Additional arguments can be controlled with `fc_method`, `aggregate_method`, `pvalue.threshold`, `logfold.threshold`, `abundance.threshold`, see `$foldchange()` for more information.

The above functions are called in order, only significant contrasts are further used to create composition and volcano plots. The `autoFlow` is intended as a first screening of certain contrasts and extra analysis, such as the use of a permutation design are not supported and should be applied by the user.

The `report` argument controls if an `rmarkdown` document is created, and with `filename` the output filename can be specified.

Usage:

```
omics$autoFlow(
  feature_contrast = "FEATURE_ID",
  feature_filter = NULL,
  feature_ranks = NULL,
  distance_metrics = c("bray"),
  distmat = NULL,
  weighted = TRUE,
  fc_method = "log",
  aggregate_method = "median",
  pvalue.threshold = 0.05,
  logfold.threshold = 0.06,
  abundance.threshold = 0,
  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, see `$feature_merge()` (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 in `ordination()` (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 to be used in `$ordination()` (default: `NULL`).

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

`fc_method` A character to choose the method in `$foldchange()` (default: `"identity"`).

- `"identity"` Computes fold-change via `log2(condition_A) - log2(condition_B)`, and will handle zero's to prevent `Inf` values. Proportional data is also supported and will be automatically detected.
- `"log"` Computes fold-change via `condition_A - condition_B`.

`aggregate_method` A function to aggregate the matrix values in `$foldchange()` by taking e.g. the `median` of `condition_A` and `condition_B` prior to substraction, or in the case of `method = "identity"` to take the median of the fold-change `log2(median(A)) - log2(median(B))` (default: `median`).

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

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

`abundance.threshold` A numeric value used as an abundance threshold to size the scatter dots based on their mean abundance in `$foldchange()` (default: `0`).

`perm` A wholenumber, number of permutations to compare against the null hypothesis of adonis2 or anosim in `$ordination()` (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: if `report = TRUE` a HTML report is generated, otherwise a list containing:

plots
- `rankstat_plot` as a ggplot.
- `alpha_div_plots` as a list of ggplot.
- `pcoa_plots` as a nested matrix of lists with ggplot (`contrasts` x `distance_metrics`).
- `composition_plots` as a nested matrix of lists with ggplot (`contrasts` x `feature_contrasts`).
- `Log2FC_plots` as a nested matrix of lists with ggplot (`contrasts` x `feature_contrasts`).

data
- `alpha_div_data` as a list of lists of `data` and `stats`.
- `pcoa_data` as a nested matrix of lists of `stats`, `dist_mat` and `pcs` (`contrasts` x `distance_metrics`).
- `composition_data` as a nested matrix of lists of `data` (`contrasts` x `feature_contrasts`).
- `Log2FC_data` as a nested matrix of lists of `data` (`contrasts` x `feature_contrasts`).

`omics$clone()`**:** The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `omics$new()`
## ------------------------------------------------

library("OmicFlow")

## Method 1: load from filepath
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 2: Load from data.frame or matrix
n_cols <- 5
n_rows <- 100
n_vals <- n_cols * n_rows
metadata <- data.frame("SAMPLE_ID" = paste0("Sample_", 1:n_cols))
features <- data.frame("FEATURE_ID" = paste0("protein_", 1:n_rows))
counts <- Matrix::Matrix(
 1:n_vals, nrow = n_rows, ncol = n_cols,
 dimnames = list(features$FEATURE_ID, metadata$SAMPLE_ID)
)

obj <- omics$new(
 metaData = metadata,
 featureData = features, # optional
 countData = counts
)

## Method 3: you have features and counts in a single data.frame
metadata <- data.frame(SAMPLE_ID = c("S1", "S2", "S3"))
counts <- data.frame(
 S1 = c(2,3,0),
 S2 = c(2,"",1),
 S3 = c(2,1,NA),
 proteins = c("ZEB1", "ZEB2", "MAPK")
)

obj <- omics$new(
 metaData = metadata,
 countData = counts
)

## ------------------------------------------------
## 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("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("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(groups = "treatment",
                           metric = "shannon")


## ------------------------------------------------
## Method `omics$composition()`
## ------------------------------------------------

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("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$new(
 metaData = metadata_file,
 countData = counts_file,
 featureData = features_file
)
## Method 1: using non-log transformation
obj$scale(method = "tss")

dfe <- obj$foldchange(
 feature_rank = "Genus",
 feature_merge = TRUE,
 method = "identity",
 aggregate_method = "median",
 condition.group = "treatment",
 condition_A = "tumor",
 condition_B = "healthy"
)

## Method 2: using log transformation
obj$reset()
obj$scale(method = "clr")

dfe <- obj$foldchange(
 feature_rank = "Genus",
 feature_merge = TRUE,
 method = "log",
 aggregate_method = "mean",
 condition.group = "treatment",
 condition_A = "tumor",
 condition_B = "healthy"
)
```

`ordination_plot` *Ordination plot*

Description

Creates an ordination plot from pre-computed principal components. 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 = lifecycle::deprecated(),
  groups = lifecycle::deprecated(),
  pair,
  dist_explained = NULL,
  dist_metric = NULL
)
```

Arguments

data	A data.frame or data.table with Principal Components as columns and rows as loading scores.
col_name	**[Deprecated]** This argument has been renamed to groups for more clarity.
groups	A categorical variable in data to color the groups (e.g. "treatment").
pair	A vector of character variables indicating what dimension names (e.g. c("PC1", "PC2") or c("NMDS1", "NMDS2")) in data.
dist_explained	A vector of numeric values of the percentage dissimilarity explained for the dimension pairs (default: NULL).
dist_metric	A character variable indicating what metric is used (e.g. "bray-curtis") (default: NULL).

Value

A ggplot2 object to be further modified

Examples

```
# 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,
  groups = "groups",
  pair = c("PC1", "PC2")
)

# Adding variance/dissimilarity explained.
ordination_plot(
  data = mock_data,
  groups = "groups",
```

```
  pair = c("PC1", "PC2"),
  dist_explained = c(45, 22),
  dist_metric = "bray-curtis"
)
```

`pairwise_adonis` *Pairwise adonis2 (PERMANOVA)*

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

`x`	A distance matrix in the form of dist. Obtained from a dissimilarity metric, in the case of similarity metric please use `1-dist`
`groups`	A character vector with equal length as the distance matrix.
`metadata`	A data.table or data.frame as input to the function `perm_design` (default: `NULL`).
`perm_design`	A function that takes the `metadata` and returns a permutation design with how (default: `NULL`).
`p.adjust.method`	A character variable to adjust the p-values, see p.adjust (default: `"bonferroni"`).
`perm`	A whole number to define the number of permutations in adonis2 (default: `999`).
`...`	Additional arguments passed to adonis2.

Value

A data.frame containing:

pairs combinations of group comparisons

Df Degrees of Freedom

SumsOfSqs The sums of squares (centroid) of the null hypothesis H_0

F.Model The F-test of the null hypothesis H_0

R2 The R squared of the null hypothesis H_0

p.value The number of F^p higher than the null hypothesis divided by the total number of permutations.

p.adj The adjusted P-value based on the used `p.adjust.method`

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*

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

x	A distance matrix in the form of dist. Obtained from a dissimilarity metric, in the case of similarity metric please use 1-dist
groups	A character vector with equal length as the distance matrix.
metadata	A data.table or data.frame as input to the function perm_design (default: NULL).
perm_design	A function that takes the metadata and returns a permutation design with how (default: NULL).
p.adjust.method	A character variable to adjust the p-values, see p.adjust (default: "bonferroni").
perm	A whole number to define the number of permutations in adonis2 (default: 999).
...	Additional arguments passed to anosim.

Value

A data.frame containing:

pairs combinations of group comparisons

Df Degrees of Freedom

R2 The R squared of the null hypothesis H_0

p.value The number of F^p higher than the null hypothesis divided by the total number of permutations.

p.adj The adjusted P-value based on the used p.adjust.method

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 *Creates barplot from* pairwise_adonis *or* pairwise_anosim

Description

Creates a pairwise barplot of stats and F-test statistics 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 in data.
group_col	A column name of a categorical variable in data.
label_col	A column name of a categorical variable to label the bars in data.
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

```
# 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

OmicFlow is constructed around the main superclass omics that in turn is inherited by metagenomics and proteomics, that offer additional fields or methods. The superclass omics contains both public and private methods, only the public componenents are documented, whereas the private methods can be accessed via `proteomics$private_methods`, which is not supported.

The omics class components use Matrix and data.table in the background for fast loading and data wrangling, these data structures return the data by reference, so does the R6 class, therefore classes in OmicFlow do not create copies of the object unless called via `$copy()`.

All classes needs to be first initialised via the `$new()` method and require the `metaData` and `countData` field components. Alternatively, you can also initialise with only the `metaData` and later add the other fields via the active binding, but this will not create a back-up and is not advised! For more hands-on examples see Getting Started with OmicFlow.

Additional arguments in `proteomics$new()`: The `proteomics` class also offers the loading of a phylogenetic tree as a `phylo` class (see as.phylo) that can be supplied to `treeData`. When a `treeData` is supplied than every class field is arranged according to the tree tip labels.

Input requirements

Every class in OmicFlow requires at least a `metaData` and `countData` field to be specified. The `metaData` must contain a `SAMPLE_ID` and optionally a `SAMPLEPAIR_ID`, see section "Metadata validation" below.

The `countData` can be both a matrix or data frame. If the original contains rownames, then these will become the `FEATURE_ID` in `featureData` (automatically created), otherwise a random feature identifier is assigned. If the `countData` is a filetype or a data frame with both matching sample identifiers in `SAMPLE_ID` and other additional columns, then these will be automatically split, columns matching the `SAMPLE_ID` will be placed in the `countData`, any other columns will be placed in front of the `FEATURE_ID` column in `featureData`.

Metadata validation

Every class in OmicFlow begins by validating the sample metadata. This validation checks that sample identifiers match the sample IDs present in the abundance tables.

The validation rules are defined in JSON format in the omics class. Metadata can be supplied as either:

- A CSV or TSV file.
- A `data.table`.

In both cases, metadata must include a header row and follow a one-row-per-sample structure. For file inputs, the header must be on the first line. Columns not described below are permitted and are ignored during metadata validation.

Minimum requirement: The metadata must contain a `SAMPLE_ID` column.

- Each row must have a unique, non-empty sample identifier.
- Sample identifiers cannot contain spaces; use underscores (_) or dashes (-) instead.

Example metadata:

SAMPLE_ID	SAMPLEPAIR_ID	CONTRAST_Treatment	VARIABLE_Age
S1	P1	Drug	42
S2	P1	Placebo	36
S3	P2	Drug	51

Column types and naming rules:

Required column

Column	Type	Rules
SAMPLE_ID	string	Unique, non-empty, no spaces; one identifier per sample row

Optional standard columns

Column	Type	Rules
SAMPLEPAIR_ID	string	Optional; no spaces. Used for samples that are paired or originate from the same individual or subj

Some functions require a `SAMEPLEPAIR_ID` in the `metaData` when `paired = TRUE` is used. The user can add or remove the `SAMPLEPAIR_ID` at anytime to the `metaData` via active binding.

Pattern-based columns

Additional variables can be defined with special column-name prefixes:

- CONTRAST_...: Grouping or category labels for differential comparisons. For example, CONTRAST_Treatment may contain Drug and Placebo.
- VARIABLE_...: Numeric or string variables for statistical analysis. For example, VARIABLE_Age may contain values such as 42 and 51.

Pattern-based columns are only used in $autoFlow(). Currently, only columns with the CONTRAST_ prefix are supported.

Super class

omics -> proteomics

Active bindings

treeData A "phylo" class, see as.phylo.

Methods

Public methods:

- proteomics$new()

proteomics$new():

Usage:

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

Arguments:

countData A path to an existing file, Matrix, data.table or data.frame (default: NULL).

metaData A path to an existing file, data.table or data.frame (default: NULL).

featureData A path to an existing file, data.table or data.frame (default: NULL).

treeData A path to an existing newick file or class "phylo", see read.tree (default: NULL).

Returns: A new proteomics object.

Examples:

```
library("OmicFlow")

## Method 1: load from filepath
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
obj <- proteomics$new(
 metaData = metadata_file,
 countData = counts_file
)
```

```
## Method 2: Load from data.frame or matrix
n_cols <- 5
n_rows <- 100
n_vals <- n_cols * n_rows
metadata <- data.frame("SAMPLE_ID" = paste0("Sample_", 1:n_cols))
features <- data.frame("FEATURE_ID" = paste0("protein_", 1:n_rows))
counts <- Matrix::Matrix(
 1:n_vals, nrow = n_rows, ncol = n_cols,
 dimnames = list(features$FEATURE_ID, metadata$SAMPLE_ID)
)

obj <- proteomics$new(
 metaData = metadata,
 featureData = features, # optional
 countData = counts
)

## Method 3: you have features and counts in a single data.frame
metadata <- data.frame(SAMPLE_ID = c("S1", "S2", "S3"))
counts <- data.frame(
 S1 = c(2,3,0),
 S2 = c(2,"",1),
 S3 = c(2,1,NA),
 proteins = c("ZEB1", "ZEB2", "MAPK")
)

obj <- proteomics$new(
 metaData = metadata,
 countData = counts
)
```

Examples

```
## ------------------------------------------------
## Method `proteomics$new()`
## ------------------------------------------------

library("OmicFlow")

## Method 1: load from filepath
metadata_file <- system.file("extdata", "metadata.tsv", package = "OmicFlow")
counts_file <- system.file("extdata", "counts.tsv", package = "OmicFlow")
obj <- proteomics$new(
 metaData = metadata_file,
 countData = counts_file
)

## Method 2: Load from data.frame or matrix
n_cols <- 5
n_rows <- 100
```

```
n_vals <- n_cols * n_rows
metadata <- data.frame("SAMPLE_ID" = paste0("Sample_", 1:n_cols))
features <- data.frame("FEATURE_ID" = paste0("protein_", 1:n_rows))
counts <- Matrix::Matrix(
 1:n_vals, nrow = n_rows, ncol = n_cols,
 dimnames = list(features$FEATURE_ID, metadata$SAMPLE_ID)
)

obj <- proteomics$new(
 metaData = metadata,
 featureData = features, # optional
 countData = counts
)

## Method 3: you have features and counts in a single data.frame
metadata <- data.frame(SAMPLE_ID = c("S1", "S2", "S3"))
counts <- data.frame(
 S1 = c(2,3,0),
 S2 = c(2,"",1),
 S3 = c(2,1,NA),
 proteins = c("ZEB1", "ZEB2", "MAPK")
)

obj <- proteomics$new(
 metaData = metadata,
 countData = counts
)
```

`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, normalize = TRUE, threads = 1)
```

Arguments

`x`	A matrix, sparseMatrix or Matrix.
`tree`	A `phylo` class tree.
`weighted`	A boolean value, to use abundances (`weighted = TRUE`) or absence/presence (`weighted=FALSE`) (default: `TRUE`).
`normalize`	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`): $d(A,B) = \sum_i^n L_i |A_i - B_i|$

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, normalize=FALSE)

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

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

volcano_plot *Volcano plot*

Description

Creates a Volcano plot from the output of `foldchange` method from class omics, it plots the fold-changes on the x-axis, log10 trasnformed 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,
  abundance_col,
  feature_rank,
  pvalue.threshold = 0.05,
  logfold.threshold = 0.6,
  abundance.threshold = 0.01,
  label_A = "A",
  label_B = "B"
)
```

Arguments

`data`	A data.table.
`logfold_col`	A column name of a continuous variable.
`pvalue_col`	A column name of a continuous variable.
`abundance_col`	A column name of a continuous variable.
`feature_rank`	A character variable of the feature column.
`pvalue.threshold`	A P-value threshold (default: `0.05`).
`logfold.threshold`	A numeric value to set the fold-change threshold (default: `0.6`).
`abundance.threshold`	An abundance threshold (default: `0.01`).
`label_A`	A character to describe condition A.
`label_B`	A character to describe condition B.

Value

A ggplot2 object to be further modified.

Examples

```
# Create mock data frame
mock_volcano_data <- data.table::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 = stats::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, *37*, *42*, *51–53*
anosim, *37*, *42*, *52*, *53*
as.phylo, *19*, *21*, *55*, *57*

boxjitter_test, 2, *33*
bray, 5, *35*
brewer.pal, *7*, *33*, *35*

canberra, 6, *36*
colormap, *3*, 7, *9*, *33*, *35*
column_exists, 8
complete.cases, *29*
composition_plot, 8, *34*
cosine, 10, *35*

data.frame, *3*, *7–9*, *21*, *26*, *34*, *37*, *50*, *51*, *53*, *54*, *57*
data.table, *3*, *7–9*, *19*, *21*, *24–26*, *35*, *39*, *50*, *51*, *53–55*, *57*, *61*
dist, *5*, *6*, *11*, *13*, *16–18*, *36*, *37*, *51*, *53*, *60*
diversity, *11*, 11, *12*, *33*, *34*

estimateSizeFactors, *39*
euclidean, 13, *35*

ggplot, *9*, *33*, *34*, *37*, *39*, *42*
ggplot2, *3*, *9*, *50*, *54*, *61*

h5read, *21*
hill_taxa, *14*, 14, *15*
how, *37*, *51*, *53*

jaccard, 15, *35*
jsd, 16, *35*

log, *12*, *14*, *32*, *36*

manhattan, 17, *35*
Matrix, *5*, *6*, *10*, *12–16*, *18*, *19*, *21*, *24*, *26*, *37*, *41*, *55*, *57*, *59*
matrix, *5*, *6*, *10*, *12–16*, *18*, *19*, *37*, *39*, *59*
matrix_to_dtable, 19
metagenomics, *7*, *11*, *19*, 19, *21*, *24*, *40*, *49*, *51*, *52*, *54*, *55*, *61*
metaMDS, *37*

omics, *2*, *7*, *8*, *11*, *19*, *21*, 24, *26*, *28*, *49*, *51*, *52*, *54*, *55*, *57*, *61*
ordination_plot, *36*, 49

p.adjust, *3*, *51*, *53*
pairwise_adonis, *36*, *37*, 51, *54*
pairwise_anosim, *36*, *37*, 52, *54*
pairwise_wilcox_test, *34*
plot_pairwise_stats, *36*, 54
proteomics, *7*, *11*, *19*, *24*, *49*, *51*, *52*, *54*, *55*, 55, *57*, *61*

read.tree, *21*, *57*
row_brownforsythe, *38*
row_levene, *38*
row_wilcoxon_paired, *3*

setNames, *3*, *7*, *9*, *35*
setThreadOptions, *5*, *6*, *10*, *13*, *15*, *16*, *18*, *59*
sparseMatrix, *5*, *6*, *10–16*, *18*, *19*, *25*, *39*, *59*
specnumber, *33*

unifrac, *36*, *37*, 59

validate_json_file, *21*
volcano_plot, *38*, *39*, 61

wcmdscale, *37*
wilcox.test, *34*