Package 'OmicFlow'

September 4, 2025

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

Version 1.3.1

Date 2025-09-04

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

License MIT + file LICENSE

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

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

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

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

Imports ape, gghalves, ggpubr, ggrepel, jsonlite, jsonvalidate, magrittr, patchwork, purrr, rbiom, RColorBrewer, rhdf5, rstatix, slam, stats, tools, utils, vegan, viridis, yyjsonr, methods, ggplot2

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

NeedsCompilation no

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 2025-09-04 06:50:02 UTC

Contents

`colormap` *Color map of a variable*

Description

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

Usage

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

Arguments

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

Value

A setNames.

Examples

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

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

`column_exists` *Checks if column exists in table*

Description

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

Usage

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

Arguments

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

Value

A boolean value.

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

Value

A ggplot2 object to be further modified

Examples

```
library("ggplot2")

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

# Create a colormap
mock_palette <- c(
  GenusA = "#1f77b4",  # blue
```

```
  GenusB = "#ff7f0e",  # orange
  GenusC = "#2ca02c",  # green
  GenusD = "#d62728",  # red
  GenusE = "#9467bd"   # purple
)

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

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

composition_plot(
  data = mock_data,
  palette = mock_palette,
  feature_rank = "Genus",
  title_name = "Mock Genus Composition by Group",
  group_by = "Group"
)
```

`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 mathematical correct. This function is built into the class omics with method `alpha_diversity()` and inherited by other omics classes, such as; metagenomics and proteomics.

Usage

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

Arguments

x	A matrix or sparseMatrix.
metric	A character variable for metric; shannon, simpson or invsimpson.
normalize	A boolean variable for sample normalization by column sums.
base	Input for log to use natural logarithmic scale, log2, log10 or other.

Value

A numeric vector with type double.

See Also

diversity

Examples

```
library("Matrix")

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

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

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

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

`diversity_plot` *Diversity plot*

Description

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

Usage

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

Arguments

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

Value

A ggplot2 object to be further modified

Examples

```
library("ggplot2")

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

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

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

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

dt <- data.table::data.table(
  "values" = div,
  "treatment" = c(rep("healthy", n_col / 2), rep("tumor", n_col / 2))
)

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

diversity_plot(
 data = dt,
 values = "values",
 col_name = "treatment",
 palette = colors,
 method = "shannon",
 paired = FALSE,
 p.adjust.method = "fdr"
)
```

`foldchange` *Computes Log2(A) - Log2(B) Fold Change of (non-) paired data.*

Description

Computes (non-)paired Log2(A) - Log2(B) Fold Change. This function is built into the class omics with method `DFE()` and inherited by other omics classes, such as; metagenomics and proteomics. The function handles zero's, and doesn't return +/- infinites.

Usage

```
foldchange(
  data,
  feature_rank,
  condition_A,
```

```
  condition_B,
  condition_labels,
  paired = FALSE
)
```

Arguments

`data`	A data.table.
`feature_rank`	A character variable of the feature level (e.g. "Genus" in taxonomy).
`condition_A`	A vector of categorical characters, it is possible to specify multiple labels.
`condition_B`	A vector of categorical characters, it is possible to specify multiple labels.
`condition_labels`	A vector character wherein `condition_A` and `condition_B` are present.
`paired`	A Boolean value to perform paired or non-paired test, see wilcox.test.

Value

A data.table

Examples

```
#------------------------#
##      NON-PAIRED       ##
#------------------------#
# Load required library
library(data.table)

# Define parameters and variables
sample_ids <- c("S1_A", "S2_A", "S3_A", "S4_B", "S5_B", "S6_B")
feature_ids <- c("Feature1", "Feature2", "Feature3")

# Simulated abundance matrix (features x samples)
abundances <- matrix(
  c(
    # Feature1 (e.g. GenusA)
    100, 120, 110,  55, 60, 65,
    # Feature2 (e.g. GenusB)
    50, 65, 60,    130, 120, 125,
    # Feature3 (e.g. GenusC)
    80, 85, 90,     80, 85, 90
  ),
  nrow = 3, byrow = TRUE,
  dimnames = list(feature_ids, sample_ids)
)

# A wide table with columns as samples, rows as features
# And an additional column as the feature_rank, a column for feature comparison.
mock_data <- data.table(
  Genus = feature_ids,  # feature_rank column (e.g. "Genus")
  S1_A = abundances[ , 1],
```

```
  S2_A = abundances[ , 2],
  S3_A = abundances[ , 3],
  S4_B = abundances[ , 4],
  S5_B = abundances[ , 5],
  S6_B = abundances[ , 6]
)
print(mock_data)

# It uses substring matching, and multiple conditions can be used
res <- foldchange(
  data = mock_data,
  feature_rank = "Genus",
  condition_A = c("_A", "_B"),
  condition_B = c("_B", "_A"),

  # This can also be a column wherein, conditions A and B are present
  condition_labels = sample_ids,
  paired = FALSE
)
print(res)

#--------------------#
##      PAIRED       ##
#--------------------#
library(data.table)

# Define paired sample ids for 3 pairs:
paired_ids <- paste0("Pair", 1:3)

# Features:
feature_ids <- c("Feature1", "Feature2", "Feature3")

# Simulate abundances for each paired sample:
# For each pair, we have two samples: condition A and condition B.
# Make sure the length of condition A and condition B are the same!

# Construct the data.table with features as rows
mock_data_paired <- data.table(
  Genus = feature_ids,
  Pair1_A = c(100, 50, 80),
  Pair1_B = c(60, 130, 75),
  Pair2_A = c(120, 65, 85),
  Pair2_B = c(60, 120, 90),
  Pair3_A = c(110, 60, 90),
  Pair3_B = c(65, 125, 85)
)

res <- foldchange(
  data = mock_data_paired,
  feature_rank = "Genus",
  condition_A = c("_A", "_B"),
  condition_B = c("_B", "_A"),
```

```
  # This can also be a column wherein, conditions A and B are present
  condition_labels = names(mock_data_paired)[-1],
  paired = TRUE
)
print(res)
```

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

Value

A numeric vector with type double.

See Also

hill_taxa

Examples

```
library("Matrix")

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

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

```
col_idx <- ((positions - 1) %/% n_row) + 1

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

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

`metagenomics` *Sub-class metagenomics*

Description

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

Super class

`OmicFlow::omics` -> `metagenomics`

Public fields

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

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

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

`treeData` A path to an existing newick file or class "phylo", see read.tree.

`biomData` A path to an existing biom file or hdf5 file, see h5read.

Methods

Public methods:

- `metagenomics$new()`
- `metagenomics$print()`
- `metagenomics$reset()`

- metagenomics$removeZeros()
- metagenomics$write_biom()

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

Usage:

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

Arguments:

`countData` countData A path to an existing file or sparseMatrix.

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

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

`treeData` A path to an existing newick file or class "phylo", see read.tree.

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

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

Returns: A new `metagenomics` object.

Method `print()`**:** Displays parameters of the metagenomics object via stdout.

Usage:

```
metagenomics$print()
```

Returns: object in place

Examples:

```
taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# method 1 to call print function
taxa

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

Method `reset()`**:** Upon creation of a new `metagenomics` object a small backup of the original data is created. Since modification of the object is done by reference and duplicates are not made, it is possible to `reset` changes to the class. The methods from the abstract class omics also contains a private method to prevent any changes to the original object when using methods such as `ordination` `alpha_diversity` or `$DFE`.

Usage:

```
metagenomics$reset()
```

Returns: object in place

Examples:

```
library(ggplot2)

taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# Performs modifications
taxa$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 `removeZeros()`**:** Removes empty (zero) values by row, column and tips from the `countData` and `treeData`. This method is performed automatically during subsetting of the object.

Usage:

```
metagenomics$removeZeros()
```

Returns: object in place

Examples:

```
taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# Sample subset induces empty features
taxa$sample_subset(treatment == "tumor")

# Remove empty features from countData and treeData
taxa$removeZeros()
```

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

Usage:

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

Arguments:

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

Examples:

```
taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

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

See Also

omics

Examples

```
## ------------------------------------------------
## Method `metagenomics$print`
## ------------------------------------------------

taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# method 1 to call print function
taxa

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


## ------------------------------------------------
## Method `metagenomics$reset`
## ------------------------------------------------

library(ggplot2)

taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# Performs modifications
taxa$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 `metagenomics$removeZeros`
## ------------------------------------------------

taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

# Sample subset induces empty features
taxa$sample_subset(treatment == "tumor")

# Remove empty features from countData and treeData
taxa$removeZeros()
```

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

taxa_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
taxa <- readRDS(taxa_path)

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

`omics` *Abstract omics class*

Description

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

Details

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

Value

A list of components:

- `div` A data.frame from diversity.
- `stats` A pairwise statistics from pairwise_wilcox_test.
- `plot` A ggplot object.

A list of components:

- `data` A data.table of feature compositions.
- `palette` A setNames palette from colormap.

A list of components:

- `distmat` A distance dissimilarity in matrix format.
- `stats` A statistical test as a data.frame.
- `pcs` principal components as a data.frame.
- `scree_plot` A ggplot object.
- `anova_plot` A ggplot object.
- `scores_plot` A ggplot object.
- `dfe` A long data.table table.
- `volcano_plot` A ggplot object.

Public fields

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

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

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

`.valid_schema` Boolean value for schema validation via JSON

`.feature_id` A character, default name for the feature identifiers.

`.sample_id` A character, default name for the sample identifiers.

`.samplepair_id` A character, default name for the sample pair identifiers.

Methods

Public methods:

- `omics$new()`
- `omics$validate()`
- `omics$removeZeros()`
- `omics$removeNAs()`
- `omics$feature_subset()`
- `omics$sample_subset()`
- `omics$samplepair_subset()`
- `omics$feature_merge()`
- `omics$transform()`
- `omics$normalize()`
- `omics$rankstat()`
- `omics$alpha_diversity()`
- `omics$composition()`
- `omics$ordination()`
- `omics$DFE()`
- `omics$autoFlow()`

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

Usage:

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

Arguments:

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

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

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

Returns: A new `omics` object.

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

Acceptable column headers:

- SAMPLE_ID (required)
- SAMPLEPAIR_ID (optional)
- FEATURE_ID (optional)
- CONTRAST_ (optional), used for `autoFlow()`.
- VARIABLE_ (optional), not supported yet.

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

Usage:

```
omics$validate()
```

Returns: None

Method `removeZeros()`**:** Removes empty (zero) values by row and column from the `countData`. This method is performed automatically during subsetting of the object.

Usage:

```
omics$removeZeros()
```

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$removeZeros()
```

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

Usage:

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

Arguments:

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

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$removeNAs(column = "treatment")
```

Method `feature_subset()`**:** Feature subset (based on `featureData`), automatically applies `removeZeros()`.

Usage:

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

Arguments:

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

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$feature_subset(Genus == "Streptococcus")
```

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

Usage:

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

Arguments:

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

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$sample_subset(treatment == "tumor")
```

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

Usage:

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

Arguments:

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

Returns: object in place

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

Usage:

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

Arguments:

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

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

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

Method `transform()`**:** Performs transformation on the positive values from the `countData`.

Usage:

```
omics$transform(FUN)
```

Arguments:

`FUN` A function such as `log2`, `log`

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$transform(log2)
```

Method `normalize()`**:** Relative abundance computation by column sums on the `countData`.

Usage:

```
omics$normalize()
```

Returns: object in place

Examples:

```
obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$normalize()
```

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

Usage:

```
omics$rankstat(feature_ranks)
```

Arguments:

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

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

Returns: A ggplot object.

Examples:

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

Method `alpha_diversity()`**:** Alpha diversity based on diversity

Usage:

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

Arguments:

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

`metric` An alpha diversity metric as input to diversity.

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

`evenness` A boolean wether to divide diversity by number of species, see specnumber.

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

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

Examples:

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

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

Usage:

```
omics$composition(
  feature_rank,
  feature_filter = NULL,
  col_name = NULL,
  normalize = TRUE,
```

```
  feature_top = c(10, 15),
  Brewer.palID = "RdYlBu"
)
```

Arguments:

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

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

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

`normalize` A boolean value, whether to `normalize()` by total sample sums (Default: TRUE).

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

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

Examples:

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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 `ordination()`**:** Ordination of `countData` with statistical testing.

Usage:

```
omics$ordination(
  metric = c("bray", "jaccard", "unifrac"),
  method = c("pcoa", "nmds"),
  group_by,
  distmat = NULL,
  weighted = TRUE,
  normalize = TRUE,
  cpus = 1,
  perm = 999
)
```

Arguments:

`metric` A dissimilarity or similarity metric to be applied on the `countData`, thus far supports 'bray', 'jaccard' and 'unifrac' when a tree is provided via `treeData`, see bdiv_distmat.

`method` Ordination method, supports "pcoa" and "nmds", see wcmdscale.

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

`distmat` A custom distance matrix in either dist or Matrix format.

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

`normalize` A boolean value, whether to `normalize()` by total sample sums (Default: TRUE).

`cpus` A wholenumber, indicating the number of processes to spawn (Default: 1) in bdiv_distmat.

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

Examples:

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

Method `DFE()`**:** Differential feature expression (DFE) using the foldchange for both paired and non-paired test.

Usage:

```
omics$DFE(
  feature_rank,
  feature_filter = NULL,
  paired = FALSE,
  normalize = TRUE,
  condition.group,
  condition_A,
  condition_B,
  pvalue.threshold = 0.05,
  foldchange.threshold = 0.06,
  abundance.threshold = 0
)
```

Arguments:

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

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

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

`normalize` A boolean value, whether to `normalize()` by total sample sums (Default: TRUE).

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

`condition_A` A character value or vector of characters.

condition_B A character value or vector of characters.

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

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

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

Examples:

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

Method `autoFlow()`**:** Automated Omics Analysis based on the `metaData`, see `validate()`. For now only works with headers that start with prefix `CONTRAST_`.

Usage:

```
omics$autoFlow(
 feature_ranks = c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species"),
  feature_contrast = c("Phylum", "Family", "Genus"),
  feature_filter = c("uncultured"),
  distance_metrics = c("unifrac"),
  beta_div_table = NULL,
  alpha_div_table = NULL,
  normalize = TRUE,
  weighted = TRUE,
  pvalue.threshold = 0.05,
  perm = 999,
  cpus = 1,
  filename = paste0(getwd(), "/report.html")
)
```

Arguments:

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

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

feature_filter A character vector to filter unwanted features, default: `c("uncultured")`

distance_metrics A character vector specifying what (dis)similarity metrics to use, default `c("unifrac")`

beta_div_table A path to pre-computed distance matrix, expects tsv/csv/txt file.

alpha_div_table A path to pre-computed alpha diversity with rarefraction depth, expects tsv/csv/txt from qiime2, see read_rarefraction_qiime.

`normalize` A boolean value, whether to `normalize()` by total sample sums (Default: TRUE).

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

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

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

`cpus` Number of cores to use, only used in `ordination()` when beta_div_table is not supplied.

`filename` A character to name the HTML report, it can also be a filepath (e.g. `"/path/to/report.html"`). Default: "report.html" in your current work directory.

Returns: A report in HTML format

See Also

diversity_plot

composition_plot

ordination_plot, plot_pairwise_stats, pairwise_anosim, pairwise_adonis

volcano_plot, foldchange

Examples

```
## ------------------------------------------------
## Method `omics$removeZeros`
## ------------------------------------------------

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$removeZeros()


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

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$removeNAs(column = "treatment")


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

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$feature_subset(Genus == "Streptococcus")
```

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

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$sample_subset(treatment == "tumor")


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

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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


## ------------------------------------------------
## Method `omics$transform`
## ------------------------------------------------

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$transform(log2)


## ------------------------------------------------
## Method `omics$normalize`
## ------------------------------------------------

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

obj$normalize()


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

library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

```
plt

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

library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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


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

library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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$ordination`
## ------------------------------------------------

library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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


## ------------------------------------------------
## Method `omics$DFE`
## ------------------------------------------------
```

```
library(ggplot2)

obj_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
obj <- readRDS(obj_path)

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

`ordination_plot` *Ordination plot*

Description

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

Usage

```
ordination_plot(
  data,
  col_name,
  pair,
  dist_explained = NULL,
  dist_metric = NULL
)
```

Arguments

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

Value

A ggplot2 object to be further modified

Examples

```
library(ggplot2)

# Mock principal component scores
set.seed(123)
mock_data <- data.frame(
  SampleID = paste0("Sample", 1:10),
  PC1 = rnorm(10, mean = 0, sd = 1),
  PC2 = rnorm(10, mean = 0, sd = 1),
  groups = rep(c("Group1", "Group2"), each = 5)
)

# Basic usage
ordination_plot(
  data = mock_data,
  col_name = "groups",
  pair = c("PC1", "PC2")
)

# Adding variance/dissimilarity explained.
ordination_plot(
  data = mock_data,
  col_name = "groups",
  pair = c("PC1", "PC2"),
  dist_explained = c(45, 22),
  dist_metric = "bray-curtis"
)
```

`pairwise_adonis` *Pairwise adonis2 (PERMANOVA) computation*

Description

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

Usage

```
pairwise_adonis(x, groups, 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 (column from a table) of labels.
`p.adjust.method`	P adjust method see p.adjust
`perm`	Number of permutations to compare against the null hypothesis of adonis2 (default: `perm=999`).

Value

A data.frame of

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

See Also

adonis2

Examples

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

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

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

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

---

pairwise_anosim *Pairwise anosim (ANOSIM) computation*

Description

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

Usage

```
pairwise_anosim(x, groups, 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 vector (column from a table) of labels.
p.adjust.method	P adjust method see p.adjust
perm	Number of permutations to compare against the null hypothesis of anosim (default: `perm=999`).

Value

A data.frame of

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

See Also

anosim

Examples

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

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

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

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

`plot_pairwise_stats` *Create pairwise stats plot*

Description

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

Usage

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

Arguments

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

Value

A ggplot2 object to be further modified

Examples

```
library("ggplot2")

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

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

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

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

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

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

`proteomics` *Sub-class proteomics*

Description

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

Super class

`OmicFlow::omics` -> `proteomics`

Public fields

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

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

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

`treeData` A path to an existing newick file or class "phylo", see read.tree.

Methods

Public methods:

- proteomics$new()
- proteomics$print()
- proteomics$reset()
- proteomics$removeZeros()

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

Usage:

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

Arguments:

`countData` countData A path to an existing file or sparseMatrix.

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

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

`treeData` A path to an existing newick file or class "phylo", see read.tree.

Returns: A new `proteomics` object.

Method `print()`**:** Displays parameters of the proteomics object via stdout.

Usage:

```
proteomics$print()
```

Returns: object in place

Examples:

```
prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# method 1 to call print function
prot

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

Method `reset()`**:** Upon creation of a new `proteomics` object a small backup of the original data is created. Since modification of the object is done by reference and duplicates are not made, it is possible to `reset` changes to the class. The methods from the abstract class omics also contains a private method to prevent any changes to the original object when using methods such as `ordination` `alpha_diversity` or `$DFE`.

Usage:

```
proteomics$reset()
```

Returns: object in place

Examples:

```
prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# Performs modifications
prot$transform(log2)

# resets
prot$reset()
```

Method `removeZeros()`**:** Removes empty (zero) values by row, column and tips from the `countData` and `treeData`. This method is performed automatically during subsetting of the object.

Usage:

```
proteomics$removeZeros()
```

Returns: object in place

Examples:

```
prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# Sample subset induces empty features
prot$sample_subset(treatment == "tumor")

# Remove empty features from countData and treeData
prot$removeZeros()
```

See Also

omics

Examples

```
## ------------------------------------------------
## Method `proteomics$print`
## ------------------------------------------------

prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# method 1 to call print function
prot

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


## ------------------------------------------------
## Method `proteomics$reset`
```

```
## ------------------------------------------------


prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# Performs modifications
prot$transform(log2)

# resets
prot$reset()


## ------------------------------------------------
## Method `proteomics$removeZeros`
## ------------------------------------------------

prot_path <- system.file("extdata", "mock_taxa.rds", package = "OmicFlow", mustWork = TRUE)
prot <- readRDS(prot_path)

# Sample subset induces empty features
prot$sample_subset(treatment == "tumor")

# Remove empty features from countData and treeData
prot$removeZeros()
```

`read_rarefraction_qiime` *Loads a rarefied alpha diversity table from Qiime2*

Description

Parses a QIIME2 table of rarefied data into a data.table as input to diversity_plot

Usage

```
read_rarefraction_qiime(filepath)
```

Arguments

`filepath`	A character value, filename or filepath to existing file.

Value

A data.table.

sparse_to_dtable *Converting a sparse matrix to data.table*

Description

Wrapper function that converts a sparseMatrix to data.table

Usage

```
sparse_to_dtable(sparsemat)
```

Arguments

sparsemat	A sparseMatrix class.

Value

A data.table class.

volcano_plot *Volcano plot*

Description

Creates a Volcano plot from the output of foldchange, it plots the foldchanges 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,
  feature_rank,
  abundance_col,
  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.
`feature_rank`	A character variable of the feature column.
`abundance_col`	A column name of a continuous variable.
`pvalue.threshold`	A P-value threshold (default: 0.05).
`logfold.threshold`	A Log2(A/B) 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

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

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

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

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

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

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

volcano_plot(
  data = mock_volcano_data,
```

```
  logfold_col = "log2FC",
  pvalue_col = "pvalue",
  abundance_col = "rel_abun",
  feature_rank = "Feature",
)
```

Index