Package 'microeco'

May 17, 2022

Type Package

Title Microbial Community Ecology Data Analysis

Version 0.9.0

Author Chi Liu [aut, cre],
Felipe R. P. Mansoldo [ctb],
Umer Zeeshan Ijaz [ctb],
Chenhao Li [ctb],
Yang Cao [ctb],
Minjie Yao [ctb],
Xiangzhen Li [ctb]

Maintainer Chi Liu <liuchi0426@126.com>

Description A series of statistical and plotting approaches in microbial community ecology based on the R6 class. The classes are designed for data preprocessing, taxa abundance plotting, alpha diversity analysis, beta diversity analysis, differential abundance test, null model analysis, network analysis, machine learning, environmental data analysis and functional analysis.

URL https://github.com/ChiLiubio/microeco

Depends R (>= 3.5.0)

Imports R6, stats, ape, vegan, rlang, data.table, magrittr, dplyr, tibble, scales, grid, ggplot2, RColorBrewer, reshape2

Suggests GUniFrac, MASS, ggpubr, randomForest, ggdendro, ggrepel, agricolae, gridExtra, picante, pheatmap, igraph, rgexf, mice

License GPL-3

LazyData true

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2022-05-17 09:10:16 UTC

RoxygenNote 7.1.2

R topics documented:

`clone` *Copy an R6 class object completely*

Description

Copy an R6 class object completely

Usage

```
clone(x, deep = TRUE)
```

Arguments

x	R6 class object
deep	default TRUE; deep copy

Value

identical but unrelated R6 object.

Examples

```
data("dataset")
clone(dataset)
```

---

color_palette_20 *A color palette for 20 elements.*

Description

This is one palette option for users who have <= 20 elements to plot. The palletes of RColorBrewer package provide at most 12 discrete colors, such as "Set3" and "Paired". This palette is adapted from D3.js library and can be used as a supplement.

Usage

```
color_palette_20
```

Format

An object of class `character` of length 20.

---

dataset *The dataset in the microeco package*

Description

The dataset is structured with microtable class for the demonstration of examples and tutorials.

Usage

```
data(dataset)
```

Format

An R6 class object

Details

- sample_table: sample information table
- otu_table: species-community abundance table
- tax_table: taxonomic table
- phylo_tree: phylogenetic tree
- taxa_abund: taxa abundance list with several tables for Phylum...Genus
- alpha_diversity: alpha diversity table
- beta_diversity: list with several beta diversity distance matrix

---

dropallfactors *Remove all factors in a data frame*

Description

Remove all factors in a data frame

Usage

```
dropallfactors(x, unfac2num = FALSE, char2num = FALSE)
```

Arguments

x	data frame
unfac2num	default FALSE; whether try to convert all character to numeric; if FALSE, only try to convert column with factor attribute. Note that this can only transform the columns that may be transformed to numeric without using factor.
char2num	default FALSE; whether force all the character to be numeric class by using factor as an intermediate.

Value

data frame without factor

Examples

```
data("taxonomy_table_16S")
taxonomy_table_16S[, 1] <- as.factor(taxonomy_table_16S[, 1])
str(dropallfactors(taxonomy_table_16S))
```

env_data_16S *The environmental factors for the 16S dataset in the microeco package*

Description

The environmental factors for the 16S dataset in the microeco package

Usage

```
data(env_data_16S)
```

fungi_func_FungalTraits *The FungalTraits database for fungi trait identification in the microeco package*

Description

The FungalTraits database for fungi trait identification in the microeco package

Usage

```
data(fungi_func_FungalTraits)
```

fungi_func_FUNGuild *The FUNGuild database for fungi trait identification in the microeco package*

Description

The FUNGuild database for fungi trait identification in the microeco package

Usage

```
data(fungi_func_FUNGuild)
```

`microtable` *Create microtable object to store and manage all the basic files.*

Description

This class is a wrapper for a series of operations on the original files and the basic manipulations, including the microtable object creation, data reduction, data rarefaction based on Paul et al. (2013) <doi:10.1371/journal.pone.0061217>, taxa abundance calculation, alpha and beta diversity calculation based on the An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035> and Lozupone et al. (2005) <doi:10.1128/AEM.71.12.8228–8235.2005> and other basic operations.
The tutorial website: https://chiliubio.github.io/microeco_tutorial/ or http://chiliubio.gitee.io/microeco_tutorial/

Format

microtable.

Methods

Public methods:

- `microtable$new()`
- `microtable$print()`
- `microtable$filter_pollution()`
- `microtable$rarefy_samples()`
- `microtable$tidy_dataset()`
- `microtable$add_rownames2taxonomy()`
- `microtable$cal_abund()`
- `microtable$save_abund()`
- `microtable$sample_sums()`
- `microtable$taxa_sums()`
- `microtable$sample_names()`
- `microtable$taxa_names()`
- `microtable$rename_taxa()`
- `microtable$merge_samples()`
- `microtable$merge_taxa()`
- `microtable$cal_alphadiv()`
- `microtable$save_alphadiv()`
- `microtable$cal_betadiv()`
- `microtable$save_betadiv()`
- `microtable$clone()`

Method `new()`:

Usage:

```
microtable$new(
  otu_table,
  sample_table = NULL,
  tax_table = NULL,
  phylo_tree = NULL,
  rep_fasta = NULL,
  auto_tidy = FALSE
)
```

Arguments:

`otu_table` data.frame; necessary; The feature abundance table, rows are features, e.g. species, cols are samples.

`sample_table` data.frame; default NULL; The sample information table, rows are samples, cols are sample metadata; If not provided, the function can generate a table automatically according to the sample names in otu_table.

`tax_table` data.frame; default NULL; The taxonomic information table, rows are species, cols are taxonomic classes.

`phylo_tree` phylo; default NULL; The phylogenetic tree; use read.tree function in ape package for input.

`rep_fasta` list or DNAStringSet; default NULL; The representative sequences; use read.fasta function in seqinr package or readDNAStringSet function in Biostrings package for input.

`auto_tidy` default FALSE; Whether trim the files in dataset automatically. If TRUE, all other operations that

Returns: an object of class "microtable" with the following components:

`sample_table` The sample information table.

`otu_table` The OTU table.

`tax_table` The taxonomic table.

`phylo_tree` The phylogenetic tree.

`rep_fasta` The representative sequence.

`taxa_abund` default NULL; use cal_abund function to calculate.

`alpha_diversity` default NULL; use cal_alphadiv function to calculate.

`beta_diversity` default NULL; use cal_betadiv function to calculate.

Examples:

```
data(otu_table_16S)
data(taxonomy_table_16S)
data(sample_info_16S)
data(phylo_tree_16S)
dataset <- microtable$new(otu_table = otu_table_16S)
dataset <- microtable$new(sample_table = sample_info_16S, otu_table = otu_table_16S,
  tax_table = taxonomy_table_16S, phylo_tree = phylo_tree_16S)
# trim the files in the dataset
dataset$tidy_dataset()
```

Method `print()`**:** Print the microtable object.

Usage:

```
microtable$print()
```

Method `filter_pollution()`**:** Filter the taxa considered as pollution from tax_table. This operation will remove any line of the tax_table containing any the word in taxa parameter regardless of word case.

Usage:

```
microtable$filter_pollution(taxa = c("mitochondria", "chloroplast"))
```

Arguments:

`taxa` default: c("mitochondria", "chloroplast"); filter mitochondria and chloroplast, or others as needed.

Returns: None

Examples:

```
dataset$filter_pollution(taxa = c("mitochondria", "chloroplast"))
```

Method `rarefy_samples()`**:** Rarefy communities to make all samples have same species number. See also `rrarefy` for the alternative method.

Usage:

```
microtable$rarefy_samples(sample.size = NULL, rngseed = 123, replace = TRUE)
```

Arguments:

`sample.size` default:NULL; species number, If not provided, use minimum number of all samples.

`rngseed` random seed; default: 123.

`replace` default: TRUE; See `sample` for the random sampling.

Returns: None; rarefied dataset.

Examples:

```
\donttest{
dataset$rarefy_samples(sample.size = min(dataset$sample_sums()), replace = TRUE)
}
```

Method `tidy_dataset()`**:** Tidy the object of microtable Class. Trim files in the object to make taxa and samples consistent across all files in the object. So the results are intersections.

Usage:

```
microtable$tidy_dataset(main_data = FALSE)
```

Arguments:

`main_data` default FALSE; if TRUE, only basic files in microtable object is trimmed. Otherwise, all files, including taxa_abund, alpha_diversity and beta_diversity, are all trimed.

Returns: None, Object of microtable itself cleaned up.

Examples:

```
dataset$tidy_dataset(main_data = TRUE)
```

Method `add_rownames2taxonomy()`**:** Add the rownames of tax_table as the last column of tax_table. This is especially useful when the rownames of tax_table are required as a taxonomic level for the following taxa_abund calculation and biomarker idenfification.

Usage:

```
microtable$add_rownames2taxonomy(use_name = "OTU")
```

Arguments:

`use_name` default "OTU"; The column name used in the tax_table.

Returns: new tax_table stored in object.

Examples:

```
\donttest{
dataset$add_rownames2taxonomy()
}
```

Method `cal_abund()`**:** Calculate the taxonomic abundance at each taxonomic rank.

Usage:

```
microtable$cal_abund(
  select_cols = NULL,
  rel = TRUE,
  split_group = FALSE,
  split_by = "&&",
  split_column = NULL
)
```

Arguments:

`select_cols` default NULL; numeric vector or character vector of colnames of tax_table; used to select columns to merge and calculate abundances. This is very useful if there are commented columns or some columns with multiple structure that cannot be used directly.

`rel` default TRUE; if TRUE, relative abundance is used; if FALSE, absolute abundance will be summed.

`split_group` default FALSE; if TRUE, split the rows to multiple rows according to one or more columns in tax_table. Very useful when multiple mapping info exist.

`split_by` default "&&"; Separator delimiting collapsed values; only useful when split_group == TRUE; see sep in separate_rows function.

`split_column` default NULL; character vector or list; only useful when split_group == TRUE; character vector: fixed column or columns used for the splitting in tax_table in each abundance calculation; list: containing more character vectors to assign the column names to each calculation, such as list(c("Phylum"), c("Phylum", "Class")).

Returns: taxa_abund in object.

Examples:

```
\donttest{
dataset$cal_abund()
}
```

Method `save_abund()`**:** Save taxonomic abundance to the computer local place.

Usage:

```
microtable$save_abund(dirpath = "taxa_abund")
```

Arguments:

`dirpath` default "taxa_abund"; directory name to save the taxonomic abundance files.

Examples:

```
\dontrun{
dataset$save_abund(dirpath = "taxa_abund")
}
```

Method `sample_sums()`**:** Sum the species number for each sample.

Usage:

```
microtable$sample_sums()
```

Returns: species number of samples.

Examples:

```
\donttest{
dataset$sample_sums()
}
```

Method `taxa_sums()`**:** Sum the species number for each taxa.

Usage:

```
microtable$taxa_sums()
```

Returns: species number of taxa.

Examples:

```
\donttest{
dataset$taxa_sums()
}
```

Method `sample_names()`**:** Show sample names.

Usage:

```
microtable$sample_names()
```

Returns: sample names.

Examples:

```
\donttest{
dataset$sample_names()
}
```

Method `taxa_names()`**:** Show taxa names of tax_table.

Usage:

```
microtable$taxa_names()
```

Returns: taxa names.

Examples:

```
\donttest{
dataset$taxa_names()
}
```

Method `rename_taxa()`**:** Rename the taxa, including the rownames of otu_table, rownames of tax_table, tip labels of phylogenetic tree and representative sequences.

Usage:

```
microtable$rename_taxa(newname_prefix = "ASV_")
```

Arguments:

`newname_prefix` default "ASV_"; the prefix of new names; new names will be newname_prefix + numbers according to the rowname order of otu_table.

Returns: renamed dataset.

Examples:

```
\donttest{
dataset$rename_taxa()
}
```

Method `merge_samples()`**:** Merge samples according to specific group to generate a new microtable.

Usage:

```
microtable$merge_samples(use_group)
```

Arguments:

`use_group` the group column in sample_table.

Returns: a new merged microtable object.

Examples:

```
\donttest{
dataset$merge_samples(use_group = "Group")
}
```

Method `merge_taxa()`**:** Merge taxa according to specific taxonomic rank to generate a new microtable.

Usage:

```
microtable$merge_taxa(taxa = "Genus")
```

Arguments:

`taxa` the specific rank in tax_table.

Returns: a new merged microtable object.

Examples:

```
\donttest{
dataset$merge_taxa(taxa = "Genus")
}
```

Method `cal_alphadiv()`**:** Calculate alpha diversity in microtable object.

Usage:

```
microtable$cal_alphadiv(measures = NULL, PD = FALSE)
```

Arguments:

`measures` default NULL; one or more indexes from "Observed", "Coverage", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson", "Fisher", "PD"; If null, use all those measures.

`PD` TRUE or FALSE, whether phylogenetic tree should be calculated, default FALSE.

Returns: alpha_diversity stored in object.

Examples:

```
\donttest{
dataset$cal_alphadiv(measures = NULL, PD = FALSE)
class(dataset$alpha_diversity)
}
```

Method `save_alphadiv()`**:** Save alpha diversity table to the computer.

Usage:

```
microtable$save_alphadiv(dirpath = "alpha_diversity")
```

Arguments:

`dirpath` default "alpha_diversity"; directory name to save the alpha_diversity.csv file.

Method `cal_betadiv()`**:** Calculate beta diversity in microtable object, including Bray-Curtis, Jaccard, and UniFrac. See An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035> and Lozupone et al. (2005) <doi:10.1128/AEM.71.12.8228–8235.2005>.

Usage:

```
microtable$cal_betadiv(method = NULL, unifrac = FALSE, binary = FALSE, ...)
```

Arguments:

`method` default NULL; a character vector with one or more elements; If default, "bray" and "jaccard" will be used; see `vegdist` function and method parameter in vegan package.

`unifrac` default FALSE; TRUE or FALSE, whether unifrac index should be calculated.

`binary` default FALSE; TRUE is used for jaccard and unweighted unifrac; optional for other indexes.

`...` parameters passed to `vegdist` function.

Returns: beta_diversity stored in object.

Examples:

```
\donttest{
dataset$cal_betadiv(unifrac = FALSE)
class(dataset$beta_diversity)
}
```

Method `save_betadiv()`**:** Save beta diversity matrix to the computer.

Usage:

```
microtable$save_betadiv(dirpath = "beta_diversity")
```

Arguments:

`dirpath` default "beta_diversity"; directory name to save the beta diversity matrix files.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `microtable$new`
## ------------------------------------------------

data(otu_table_16S)
data(taxonomy_table_16S)
data(sample_info_16S)
data(phylo_tree_16S)
dataset <- microtable$new(otu_table = otu_table_16S)
dataset <- microtable$new(sample_table = sample_info_16S, otu_table = otu_table_16S,
  tax_table = taxonomy_table_16S, phylo_tree = phylo_tree_16S)
# trim the files in the dataset
dataset$tidy_dataset()

## ------------------------------------------------
## Method `microtable$filter_pollution`
## ------------------------------------------------

dataset$filter_pollution(taxa = c("mitochondria", "chloroplast"))

## ------------------------------------------------
## Method `microtable$rarefy_samples`
## ------------------------------------------------


dataset$rarefy_samples(sample.size = min(dataset$sample_sums()), replace = TRUE)


## ------------------------------------------------
## Method `microtable$tidy_dataset`
## ------------------------------------------------

dataset$tidy_dataset(main_data = TRUE)

## ------------------------------------------------
## Method `microtable$add_rownames2taxonomy`
## ------------------------------------------------


dataset$add_rownames2taxonomy()


## ------------------------------------------------
## Method `microtable$cal_abund`
## ------------------------------------------------


dataset$cal_abund()
```

```
## ------------------------------------------------
## Method `microtable$save_abund`
## ------------------------------------------------

## Not run:
dataset$save_abund(dirpath = "taxa_abund")

## End(Not run)

## ------------------------------------------------
## Method `microtable$sample_sums`
## ------------------------------------------------


dataset$sample_sums()


## ------------------------------------------------
## Method `microtable$taxa_sums`
## ------------------------------------------------


dataset$taxa_sums()


## ------------------------------------------------
## Method `microtable$sample_names`
## ------------------------------------------------


dataset$sample_names()


## ------------------------------------------------
## Method `microtable$taxa_names`
## ------------------------------------------------


dataset$taxa_names()


## ------------------------------------------------
## Method `microtable$rename_taxa`
## ------------------------------------------------


dataset$rename_taxa()


## ------------------------------------------------
## Method `microtable$merge_samples`
## ------------------------------------------------
```

```
dataset$merge_samples(use_group = "Group")


## ------------------------------------------------
## Method `microtable$merge_taxa`
## ------------------------------------------------


dataset$merge_taxa(taxa = "Genus")


## ------------------------------------------------
## Method `microtable$cal_alphadiv`
## ------------------------------------------------


dataset$cal_alphadiv(measures = NULL, PD = FALSE)
class(dataset$alpha_diversity)


## ------------------------------------------------
## Method `microtable$cal_betadiv`
## ------------------------------------------------


dataset$cal_betadiv(unifrac = FALSE)
class(dataset$beta_diversity)
```

`otu_table_16S` *The OTU table of the 16S dataset in the microeco package*

Description

The OTU table of the 16S dataset in the microeco package

Usage

```
data(otu_table_16S)
```

`otu_table_ITS` *The OTU table of the ITS dataset in the microeco package*

Description

The OTU table of the ITS dataset in the microeco package

Usage

```
data(otu_table_ITS)
```

`phylo_tree_16S` *The phylogenetic tree of 16S dataset in the microeco package*

Description

The phylogenetic tree of 16S dataset in the microeco package

Usage

```
data(phylo_tree_16S)
```

`prok_func_FAPROTAX` *The modified FAPROTAX trait database in the microeco package*

Description

The modified FAPROTAX trait database in the microeco package

Usage

```
data(prok_func_FAPROTAX)
```

`prok_func_NJC19_list` *The modified NJC19 database in the microeco package*

Description

The modified NJC19 database in the microeco package

Usage

```
data(prok_func_NJC19_list)
```

rep_fasta_16S *The fasta file of 16S dataset used in tax4fun2 method.*

Description

See the document of microtable class for more details. This file is with read.fasta function in seqinr package.

Usage

```
data(rep_fasta_16S)
```

sample_info_16S *The sample information of 16S dataset in the microeco package*

Description

The sample information of 16S dataset in the microeco package

Usage

```
data(sample_info_16S)
```

sample_info_ITS *The sample information of ITS dataset in the microeco package*

Description

The sample information of ITS dataset in the microeco package

Usage

```
data(sample_info_ITS)
```

Tax4Fun2_KEGG *The KEGG data files used in the cal_tax4fun2 function of trans_func class.*

Description

The KEGG data files used in the cal_tax4fun2 function of trans_func class.

Usage

```
data(Tax4Fun2_KEGG)
```

taxonomy_table_16S *The taxonomic information of 16S dataset in the microeco package*

Description

The taxonomic information of 16S dataset in the microeco package

Usage

```
data(taxonomy_table_16S)
```

taxonomy_table_ITS *The taxonomic information of ITS dataset in the microeco package*

Description

The taxonomic information of ITS dataset in the microeco package

Usage

```
data(taxonomy_table_ITS)
```

tidy_taxonomy *Clean up the taxonomic table to make taxonomic assignments consistent.*

Description

Clean up the taxonomic table to make taxonomic assignments consistent.

Usage

```
tidy_taxonomy(
  taxonomy_table,
  column = "all",
  pattern = c(".*uncultur.*", ".*unknown.*", ".*unidentif.*", ".*unclassified.*",
   ".*No blast hit.*", ".*sp\\.$", ".*metagenome.*", ".*cultivar.*", ".*archaeon$",
    "__synthetic.*", ".*\\sbacterium$", ".*bacterium\\s.*"),
  replacement = "",
  ignore.case = TRUE,
  na_fill = ""
)
```

Arguments

taxonomy_table	a data.frame with taxonomic information.
column	default "all"; "all" or a number; 'all' represents cleaning up all the columns; a number represents cleaning up this column.
pattern	default see the function parameter; the characters (regular expression) to be cleaned up or replaced; cleaned up when parameter replacement = "", replaced when parameter replacement has something; Note that the capital and small letters are not distinguished.
replacement	default ""; the characters used to replace the character in pattern parameter.
ignore.case	default TRUE; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.
na_fill	default ""; used to replace the NA.

Format

data.frame object.

Value

taxonomic table.

Examples

```
data("taxonomy_table_16S")
tidy_taxonomy(taxonomy_table_16S)
```

---

trans_abund *Create trans_abund object to transform taxonomic abundance for plotting.*

Description

This class is a wrapper for the taxonomic abundance transformations and plotting. The transformed data style is the long-format for ggplot2 plotting. The plotting methods include bar plot, boxplot, heatmap and pie chart.

Methods

Public methods:

- trans_abund$new()
- trans_abund$plot_bar()
- trans_abund$plot_heatmap()
- trans_abund$plot_box()
- trans_abund$plot_line()
- trans_abund$plot_pie()

- trans_abund$print()
- trans_abund$clone()

Method `new()`**:**

Usage:

```
trans_abund$new(
  dataset = NULL,
  taxrank = "Phylum",
  show = 0,
  ntaxa = 10,
  groupmean = NULL,
  delete_full_prefix = TRUE,
  delete_part_prefix = FALSE,
  prefix = NULL,
  use_percentage = TRUE,
  input_taxaname = NULL
)
```

Arguments:

`dataset` default NULL; the object of `microtable` class.

`taxrank` default "Phylum"; taxonomic rank.

`show` default 0; the relative abundance threshold used for filtering.

`ntaxa` default 10; how many taxa will be used, ordered by abundance from high to low; this parameter does not conflict with the parameter show; both can be used.

`groupmean` default NULL; calculating mean abundance for each group; select a group column name in sample_table.

`delete_full_prefix` default TRUE; whether delete both the prefix of taxonomy and the character in front of them.

`delete_part_prefix` default FALSE; whether only delete the prefix of taxonomy.

`prefix` default NULL; character string; can be used when delete_full_prefix = T or delete_part_prefix = T; default NULL reprensents using the "letter+__", e.g. "k__" for Phylum level; Please alter this parameter when the prefix is not standard.

`use_percentage` default TRUE; show the abundance percentage.

`input_taxaname` default NULL; character vector; if some taxa are selected, input taxa names.

Returns: data_abund stored in the object for plotting.

Examples:

```
\donttest{
data(dataset)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 10)
}
```

Method `plot_bar()`**:** Bar plot.

Usage:

```
trans_abund$plot_bar(
  color_values = RColorBrewer::brewer.pal(12, "Paired"),
  bar_type = "full",
```

```
  others_color = "grey90",
  facet = NULL,
  facet2 = NULL,
  order_facet = NULL,
  x_axis_name = NULL,
  order_x = NULL,
  barwidth = NULL,
  use_alluvium = FALSE,
  clustering = FALSE,
  facet_color = "grey95",
  strip_text = 11,
  legend_text_italic = FALSE,
  xtext_type_hor = TRUE,
  xtext_size = 10,
  xtext_keep = TRUE,
  xtitle_keep = TRUE,
  ytitle_size = 17,
  ylab_title = NULL
)
```

Arguments:

`color_values` default RColorBrewer::brewer.pal(12, "Paired"); colors palette for the plotting.

`bar_type` default "full"; "full" or "notfull"; if full, the total abundance sum to 1 or 100 percentage.

`others_color` default "grey90"; the color for "others" taxa.

`facet` default NULL; if using facet, providing a group column name of sample_table, such as, "Group".

`facet2` default NULL; the second facet, used with facet parameter together; facet2 should have a finer scale; use this parameter, please first install package ggh4x using install.packages("ggh4x")

`order_facet` NULL; vector; used to order the facet, such as, c("Group1", "Group3", "Group2").

`x_axis_name` NULL; a character string; a column name of sample_table used to show the sample names in x axis.

`order_x` default NULL; vector; used to order the sample names in x axis; must be the samples vector, such as, c("S1", "S3", "S2").

`barwidth` default NULL; bar width, see width in `geom_bar`.

`use_alluvium` default FALSE; whether add alluvium plot

`clustering` default FALSE; whether order samples by the clustering

`facet_color` default "grey95"; facet background color.

`strip_text` default 11; facet text size.

`legend_text_italic` default FALSE; whether use italic in legend.

`xtext_type_hor` default TRUE; x axis text horizontal, if FALSE; text slant.

`xtext_size` default 10; x axis text size.

`xtext_keep` default TRUE; whether retain x text.

`xtitle_keep` default TRUE; whether retain x title.

`ytitle_size` default 17; y axis title size.

`ylab_title` default NULL; y axis title.

Returns: ggplot2 plot.

Examples:

```
\donttest{
t1$plot_bar(facet = "Group", xtext_keep = FALSE)
}
```

Method `plot_heatmap()`**:** Plot the heatmap.

Usage:

```
trans_abund$plot_heatmap(
  color_values = rev(RColorBrewer::brewer.pal(n = 11, name = "RdYlBu")),
  facet = NULL,
  order_facet = NULL,
  x_axis_name = NULL,
  order_x = NULL,
  withmargin = TRUE,
  plot_numbers = FALSE,
  plot_text_size = 4,
  plot_breaks = NULL,
  margincolor = "white",
  plot_colorscale = "log10",
  min_abundance = 0.01,
  max_abundance = NULL,
  strip_text = 11,
  xtext_size = 10,
  ytext_size = 11,
  xtext_keep = TRUE,
  xtitle_keep = TRUE,
  grid_clean = TRUE,
  xtext_type_hor = TRUE,
  pheatmap = FALSE,
  ...
)
```

Arguments:

`color_values` default rev(RColorBrewer::brewer.pal(n = 11, name = "RdYlBu")); colors palette for the plotting.

`facet` default NULL; a character string; if using facet, provide a column name in sample_table, such as "Group".

`order_facet` NULL; vector; used to order the facet, such as, c("Group1", "Group3", "Group2").

`x_axis_name` NULL; a character string; a column name of sample_table used to show the sample names in x axis.

`order_x` default NULL; vector; used to order the sample names in x axis; must be the samples vector, such as, c("S1", "S3", "S2").

`withmargin` default TRUE; whether retain the tile margin.

`plot_numbers` default FALSE; whether plot the number in heatmap.

`plot_text_size` default 4; If plot_numbers TRUE, text size in plot.

`plot_breaks` default NULL; The legend breaks.

`margincolor` default "white"; If withmargin TRUE, use this as the margin color.

`plot_colorscale` default "log10"; color scale.

`min_abundance` default .01; the minimum abundance percentage in plot.

`max_abundance` default NULL; the maximum abundance percentage in plot, NULL reprensent the max percentage.

`strip_text` default 11; facet text size.

`xtext_size` default 10; x axis text size.

`ytext_size` default 11; y axis text size.

`xtext_keep` default TRUE; whether retain x text.

`xtitle_keep` default TRUE; whether retain x title.

`grid_clean` default TRUE; whether remove grid lines.

`xtext_type_hor` default TRUE; x axis text horizontal, if FALSE; text slant.

`pheatmap` default FALSE; whether use pheatmap package to plot the heatmap.

`...` paremeters pass to pheatmap when pheatmap = TRUE.

Returns: ggplot2 plot or grid plot based on pheatmap.

Examples:

```
\donttest{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 40)
t1$plot_heatmap(facet = "Group", xtext_keep = FALSE, withmargin = FALSE)
}
```

Method `plot_box()`**:** Box plot.

Usage:

```
trans_abund$plot_box(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  group = NULL,
  show_point = FALSE,
  point_color = "black",
  point_size = 3,
  point_alpha = 0.3,
  plot_flip = FALSE,
  boxfill = TRUE,
  middlecolor = "grey95",
  middlesize = 1,
  xtext_type_hor = FALSE,
  xtext_size = 10,
  xtext_keep = TRUE,
  xtitle_keep = TRUE,
  ytitle_size = 17,
  ...
)
```

Arguments:

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette for the plotting.

`group` default NULL; a column name of sample table to show abundance across groups.

`show_point` default FALSE; whether show points in plot.

point_color default "black"; If show_point TRUE; use the color

point_size default 3; If show_point TRUE; use the size

point_alpha default .3; If show_point TRUE; use the transparency.

plot_flip default FALSE; Whether rotate plot.

boxfill default TRUE; Whether fill the box with colors.

middlecolor default "grey95"; The middle line color.

middlesize default 1; The middle line size.

xtext_type_hor default TRUE; x axis text horizontal, if FALSE; text slant.

xtext_size default 10; x axis text size.

xtext_keep default TRUE; whether retain x text.

xtitle_keep default TRUE; whether retain x title.

ytitle_size default 17; y axis title size.

... parameters pass to geom_boxplot.

Returns: ggplot2 plot.

Examples:

```
\donttest{
t1$plot_box(group = "Group")
}
```

Method `plot_line()`**:** Plot the line chart.

Usage:

```
trans_abund$plot_line(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  plot_SE = TRUE,
  position = position_dodge(0.1),
  errorbar_size = 1,
  point_size = 3,
  point_alpha = 0.8,
  line_size = 0.8,
  line_alpha = 0.8,
  line_type = 1,
  xtext_type_hor = FALSE,
  xtext_size = 10,
  ytitle_size = 17
)
```

Arguments:

color_values default RColorBrewer::brewer.pal(8, "Dark2"); colors palette for the plotting.

plot_SE default TRUE; TRUE: plot the errorbar with mean±se; FALSE: plot the errorbar with mean±sd.

position default position_dodge(0.1); Position adjustment, either as a string (such as "identity"), or the result of a call to a position adjustment function.

errorbar_size default 1; errorbar size.

point_size default 3; point size for taxa.

point_alpha default 0.8; point transparency.

`line_size` default 0.8; line size.

`line_alpha` default 0.8; line transparency.

`line_type` default 1; an integer; line type.

`xtext_type_hor` default TRUE; x axis text horizontal, if FALSE; text slant.

`xtext_size` default 10; x axis text size.

`ytitle_size` default 17; y axis title size.

Returns: ggplot2 plot.

Examples:

```
\donttest{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5)
t1$plot_line(point_size = 3)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5, groupmean = "Group")
t1$plot_line(point_size = 5, errorbar_size = 1, xtext_type_hor = TRUE)
}
```

Method `plot_pie()`**:** Plot pie chart.

Usage:

```
trans_abund$plot_pie(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  facet_nrow = 1,
  strip_text = 11,
  legend_text_italic = FALSE
)
```

Arguments:

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette for the plotting.

`facet_nrow` default 1; how many rows in the plot.

`strip_text` default 11; sample title size.

`legend_text_italic` default FALSE; whether use italic in legend.

Returns: ggplot2 plot.

Examples:

```
\donttest{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_pie(facet_nrow = 1)
}
```

Method `print()`**:** Print the trans_abund object.

Usage:

```
trans_abund$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_abund$new`
## ------------------------------------------------


data(dataset)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 10)


## ------------------------------------------------
## Method `trans_abund$plot_bar`
## ------------------------------------------------


t1$plot_bar(facet = "Group", xtext_keep = FALSE)


## ------------------------------------------------
## Method `trans_abund$plot_heatmap`
## ------------------------------------------------


t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 40)
t1$plot_heatmap(facet = "Group", xtext_keep = FALSE, withmargin = FALSE)


## ------------------------------------------------
## Method `trans_abund$plot_box`
## ------------------------------------------------


t1$plot_box(group = "Group")


## ------------------------------------------------
## Method `trans_abund$plot_line`
## ------------------------------------------------


t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5)
t1$plot_line(point_size = 3)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5, groupmean = "Group")
t1$plot_line(point_size = 5, errorbar_size = 1, xtext_type_hor = TRUE)


## ------------------------------------------------
## Method `trans_abund$plot_pie`
## ------------------------------------------------
```

```
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_pie(facet_nrow = 1)
```

---

trans_alpha *Create trans_alpha object for alpha diveristy statistics and plotting.*

Description

This class is a wrapper for a series of alpha diveristy related analysis, including the statistics and plotting based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035> and Paul et al. (2013) <doi:10.1371/journal.pone.0061217>.

Methods

Public methods:

- `trans_alpha$new()`
- `trans_alpha$cal_diff()`
- `trans_alpha$plot_alpha()`
- `trans_alpha$print()`
- `trans_alpha$clone()`

Method `new()`:

Usage:

```
trans_alpha$new(dataset = NULL, group = NULL, order_x = NULL)
```

Arguments:

`dataset` the object of `microtable` Class.

`group` default NULL; the sample column used for the statistics; If provided, can return data_stat.

`order_x` default NULL; sample_table column name or a vector containg sample names; if provided, order samples by using factor.

Returns: data_alpha and data_stat stored in the object.

Examples:

```
\donttest{
data(dataset)
t1 <- trans_alpha$new(dataset = dataset, group = "Group")
}
```

Method `cal_diff()`: Test the difference of alpha diversity across groups.

Usage:

```
trans_alpha$cal_diff(
  method = c("KW", "KW_dunn", "wilcox", "t.test", "anova")[1],
  measure = NULL,
  p_adjust_method = "fdr",
  anova_set = NULL,
  ...
)
```

Arguments:

`method` default "KW"; see the following available options:

- **'KW'** KW: Kruskal-Wallis Rank Sum Test for all groups (>= 2)
- **'KW_dunn'** Dunn's Kruskal-Wallis Multiple Comparisons, see dunnTest function in FSA package
- **'wilcox'** Wilcoxon Rank Sum and Signed Rank Tests for all paired groups
- **'t.test'** Student's t-Test for all paired groups
- **'anova'** Duncan's multiple range test for anova

`measure` default NULL; a vector; if null, all indexes will be calculated; see names of microtable$alpha_diversity, e.g. Observed, Chao1, ACE, Shannon, Simpson, InvSimpson, Fisher, Coverage, PD.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of p.adjust function for available options; NULL can disuse the p value adjustment.

`anova_set` default NULL; specified group set for anova, such as 'block + N*P*K', see aov.

`...` parameters passed to kruskal.test or wilcox.test function (method = "KW") or dunnTest function of FSA package (method = "KW_dunn") or agricolae::duncan.test (method = "anova").

Returns: res_diff in object. A data.frame generally. A list for anova when anova_set is assigned. In the data frame, 'Group' column means that the group has the maximum median or mean value across the test groups; For non-parametric methods, maximum median value; For t.test, maximum mean value.

Examples:

```
\donttest{
t1$cal_diff(method = "KW")
t1$cal_diff(method = "KW_dunn")
t1$cal_diff(method = "anova")
}
```

Method `plot_alpha()`**:** Plotting the alpha diveristy.

Usage:

```
trans_alpha$plot_alpha(
  use_diff = TRUE,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  measure = "Shannon",
  group = NULL,
  add_label = "Significance",
  use_boxplot = TRUE,
  boxplot_color = TRUE,
  boxplot_add = "jitter",
  order_x_mean = FALSE,
  y_start = 1.01,
  y_increase = 0.05,
  xtext_angle = NULL,
  xtext_size = 15,
  ytitle_size = 17,
  ...
)
```

Arguments:

`use_diff` default TRUE; wheter add significance label using the result of cal_diff function, i.e. object$res_diff; This is manily designed to add post hoc test of anova or Dunn's Kruskal-Wallis Multiple Comparisons to make the label adding easy.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); color pallete for groups.

`measure` default Shannon; alpha diveristy measurement; see names of alpha_diversity of dataset, e.g., Observed, Chao1, ACE, Shannon, Simpson, InvSimpson, Fisher, Coverage, PD.

`group` default NULL; group name used for the plot.

`add_label` default "Significance"; select a colname of object$res_diff for the label text, such as 'P.adj' or 'Significance'.

`use_boxplot` default TRUE; TRUE: boxplot; FALSE: mean-se plot.

`boxplot_color` default TRUE; TRUE: use color_values, FALSE: use "black".

`boxplot_add` default "jitter"; points type, see the add parameter in ggpubr::ggboxplot.

`order_x_mean` default FALSE; whether order x axis by the means of groups from large to small.

`y_start` default 1.01; the y axis value from which to add the label; the default 1.01 means 1.01 * max(values).

`y_increase` default 0.05; the increasing y axia space to add label; the default 0.05 means 0.05 * y_start; this parameter is also used to label the letters of anova result with the fixed (1 + y_increase) * y_start space.

`xtext_angle` default NULL; number (e.g. 30) used to make x axis text generate angle.

`xtext_size` default 15; x axis text size.

`ytitle_size` default 17; y axis title size.

`...` parameters pass to ggpubr::ggboxplot function.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_alpha(measure = "Shannon", group = "Group")
}
```

Method `print()`**:** Print the trans_alpha object.

Usage:

```
trans_alpha$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

```
## Method `trans_alpha$new`
## ------------------------------------------------


data(dataset)
t1 <- trans_alpha$new(dataset = dataset, group = "Group")


## ------------------------------------------------
## Method `trans_alpha$cal_diff`
## ------------------------------------------------


t1$cal_diff(method = "KW")
t1$cal_diff(method = "KW_dunn")
t1$cal_diff(method = "anova")


## ------------------------------------------------
## Method `trans_alpha$plot_alpha`
## ------------------------------------------------


t1$plot_alpha(measure = "Shannon", group = "Group")
```

`trans_beta` *Create trans_beta object for the analysis of distance matrix of beta-diversity.*

Description

This class is a wrapper for a series of beta-diversity related analysis, including several ordination calculations and plotting based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>, group distance comparision, clustering, perMANOVA based on Anderson al. (2008) <doi:10.1111/j.1442-9993.2001.01070.pp.x> and PERMDISP. Please also cite the original paper: An et al. (2019). Soil bacterial community structure in Chinese wetlands. Geoderma, 337, 290-299.

Methods

Public methods:

- `trans_beta$new()`
- `trans_beta$cal_ordination()`
- `trans_beta$plot_ordination()`
- `trans_beta$cal_manova()`
- `trans_beta$cal_betadisper()`
- `trans_beta$cal_group_distance()`
- `trans_beta$plot_group_distance()`

- trans_beta$plot_clustering()
- trans_beta$clone()

Method `new()`**:**

Usage:

```
trans_beta$new(dataset = NULL, measure = NULL, group = NULL)
```

Arguments:

`dataset` the object of `microtable` Class.

`measure` default NULL; bray, jaccard, wei_unifrac or unwei_unifrac, or other name of matrix you add; beta diversity index used for ordination, manova or group distance.

`group` default NULL; sample group used for manova, betadisper or group distance.

Returns: parameters stored in the object.

Examples:

```
data(dataset)
t1 <- trans_beta$new(dataset = dataset, measure = "bray", group = "Group")
```

Method `cal_ordination()`**:** Ordination based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>.

Usage:

```
trans_beta$cal_ordination(
  ordination = "PCoA",
  ncomp = 3,
  trans_otu = FALSE,
  scale_species = FALSE
)
```

Arguments:

`ordination` default "PCoA"; "PCA", "PCoA" or "NMDS". PCA: principal component analysis; PCoA: principal coordinates analysis; NMDS: non-metric multidimensional scaling.

`ncomp` default 3; the returned dimensions.

`trans_otu` default FALSE; whether species abundance will be square transformed, used for PCA.

`scale_species` default FALSE; whether species loading in PCA will be scaled.

Returns: res_ordination stored in the object.

Examples:

```
t1$cal_ordination(ordination = "PCoA")
```

Method `plot_ordination()`**:** Plotting the ordination result based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.0

Usage:

```
trans_beta$plot_ordination(
  plot_type = "point",
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  plot_color = NULL,
  plot_shape = NULL,
  plot_group_order = NULL,
```

```
  add_sample_label = NULL,
  point_size = 3,
  point_alpha = 0.8,
  centroid_segment_alpha = 0.6,
  centroid_segment_size = 1,
  centroid_segment_linetype = 3,
  ellipse_chull_fill = TRUE,
  ellipse_chull_alpha = 0.1,
  ellipse_level = 0.9,
  ellipse_type = "t"
)
```

Arguments:

`plot_type` default "point"; one or more elements of "point", "ellipse", "chull" and "centroid".

- **'point'** add point
- **'ellipse'** add confidence ellipse for points of each group
- **'chull'** add convex hull for points of each group
- **'centroid'** add centroid line of each group

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette for different groups.

`shape_values` default c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14); a vector for point shape types of groups, see ggplot2 tutorial.

`plot_color` default NULL; a colname of sample_table to assign colors to different groups in plot.

`plot_shape` default NULL; a colname of sample_table to assign shapes to different groups in plot.

`plot_group_order` default NULL; a vector used to order the groups in the legend of plot.

`add_sample_label` default NULL; the column name in sample table, if provided, show the point name in plot.

`point_size` default 3; point size in plot when "point" is in plot_type.

`point_alpha` default .8; point transparency in plot when "point" is in plot_type.

`centroid_segment_alpha` default 0.6; segment transparency in plot when "centroid" is in plot_type.

`centroid_segment_size` default 1; segment size in plot when "centroid" is in plot_type.

`centroid_segment_linetype` default 3; the line type related with centroid in plot when "centroid" is in plot_type.

`ellipse_chull_fill` default TRUE; whether fill colors to the area of ellipse or chull.

`ellipse_chull_alpha` default 0.1; color transparency in the ellipse or convex hull depending on whether "ellipse" or "centroid" is in plot_type.

`ellipse_level` default .9; confidence level of ellipse when "ellipse" is in plot_type.

`ellipse_type` default "t"; ellipse type when "ellipse" is in plot_type; see type in `stat_ellipse`.

Returns: ggplot.

Examples:

```
t1$plot_ordination(plot_type = "point")
t1$plot_ordination(plot_color = "Group", plot_shape = "Group", plot_type = "point")
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "ellipse"))
```

```
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "chull"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "centroid"),
  centroid_segment_linetype = 1)
```

Method `cal_manova()`**:** Calculate perMANOVA based on Anderson al. (2008) <doi:10.1111/j.1442-9993.2001.01070.pp.x> and R vegan adonis2 function.

Usage:

```
trans_beta$cal_manova(
  manova_all = TRUE,
  manova_set = NULL,
  group = NULL,
  p_adjust_method = "fdr",
  ...
)
```

Arguments:

`manova_all` default TRUE; TRUE represents test for all the groups, i.e. the overall test; FALSE represents test for all the paired groups.

`manova_set` default NULL; other specified group set for manova, such as "Group + Type" and "Group*Type"; see also `adonis2`.

`group` default NULL; a column name of sample_table used for manova. If NULL, search group stored in the object.

`p_adjust_method` default "fdr"; p.adjust method when manova_all = FALSE; see method parameter of p.adjust function for available options.

`...` parameters passed to `adonis2` function of vegan package.

Returns: res_manova stored in object.

Examples:

```
t1$cal_manova(manova_all = TRUE)
```

Method `cal_betadisper()`**:** A wrapper for betadisper function in vegan package for multivariate homogeneity test of groups dispersions.

Usage:

```
trans_beta$cal_betadisper(...)
```

Arguments:

`...` parameters passed to `betadisper` function.

Returns: res_betadisper stored in object.

Examples:

```
t1$cal_betadisper()
```

Method `cal_group_distance()`**:** Transform sample distances within groups or between groups.

Usage:

```
trans_beta$cal_group_distance(within_group = TRUE)
```

Arguments:

`within_group` default TRUE; whether transform sample distance within groups, if FALSE, transform sample distance between any two groups.

Returns: res_group_distance stored in object.

Examples:

```
\donttest{
t1$cal_group_distance(within_group = TRUE)
}
```

Method `plot_group_distance()`**:** Plotting the distance between samples within or between groups.

Usage:

```
trans_beta$plot_group_distance(
  plot_group_order = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  distance_pair_stat = FALSE,
  hide_ns = FALSE,
  hide_ns_more = NULL,
  pair_compare_filter_match = NULL,
  pair_compare_filter_select = NULL,
  pair_compare_method = "wilcox.test",
  plot_distance_xtype = NULL
)
```

Arguments:

`plot_group_order` default NULL; a vector used to order the groups in the plot.

`color_values` colors for presentation.

`distance_pair_stat` default FALSE; whether do the paired comparisions.

`hide_ns` default FALSE; whether hide the "ns" pairs, i.e. non significant comparisions.

`hide_ns_more` default NULL; character vector; available when hide_ns = TRUE; if provided, used for the specific significance filtering, such as c("ns", "*").

`pair_compare_filter_match` default NULL; only available when hide_ns = FALSE; if provided, remove the matched groups; use the regular express to match the paired groups.

`pair_compare_filter_select` default NULL; numeric vector;only available when hide_ns = FALSE; if provided, only select those input groups. This parameter must be a numeric vector used to select the paired combination of groups. For example, pair_compare_filter_select = c(1, 3) can be used to select "CW"-"IW" and "IW"-"TW" from all the three pairs "CW"-"IW", "CW"-"TW" and "IW"-"TW" of ordered groups ("CW", "IW", "TW"). The parameter pair_compare_filter_select and pair_compare_filter_match can not be both used together.

`pair_compare_method` default wilcox.test; wilcox.test, kruskal.test, t.test or anova.

`plot_distance_xtype` default NULL; number used to make x axis text generate angle.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_group_distance(distance_pair_stat = TRUE)
t1$plot_group_distance(distance_pair_stat = TRUE, hide_ns = TRUE)
t1$plot_group_distance(distance_pair_stat = TRUE, hide_ns = TRUE, hide_ns_more = c("ns", "*"))
t1$plot_group_distance(distance_pair_stat = TRUE, pair_compare_filter_select = 3)
}
```

Method `plot_clustering()`**:** Plotting clustering result. Require ggdendro package.

Usage:

```
trans_beta$plot_clustering(
  use_colors = RColorBrewer::brewer.pal(8, "Dark2"),
  measure = NULL,
  group = NULL,
  replace_name = NULL
)
```

Arguments:

`use_colors` colors for presentation.

`measure` default NULL; beta diversity index; If NULL, using the measure when creating object

`group` default NULL; if provided, use this group to assign color.

`replace_name` default NULL; if provided, use this as label.

Returns: ggplot.

Examples:

```
t1$plot_clustering(group = "Group", replace_name = c("Saline", "Type"))
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_beta$new`
## ------------------------------------------------

data(dataset)
t1 <- trans_beta$new(dataset = dataset, measure = "bray", group = "Group")

## ------------------------------------------------
## Method `trans_beta$cal_ordination`
## ------------------------------------------------

t1$cal_ordination(ordination = "PCoA")

## ------------------------------------------------
## Method `trans_beta$plot_ordination`
## ------------------------------------------------

t1$plot_ordination(plot_type = "point")
t1$plot_ordination(plot_color = "Group", plot_shape = "Group", plot_type = "point")
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "ellipse"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "chull"))
```

```
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "centroid"),
  centroid_segment_linetype = 1)

## ------------------------------------------------
## Method `trans_beta$cal_manova`
## ------------------------------------------------

t1$cal_manova(manova_all = TRUE)

## ------------------------------------------------
## Method `trans_beta$cal_betadisper`
## ------------------------------------------------

t1$cal_betadisper()

## ------------------------------------------------
## Method `trans_beta$cal_group_distance`
## ------------------------------------------------


t1$cal_group_distance(within_group = TRUE)


## ------------------------------------------------
## Method `trans_beta$plot_group_distance`
## ------------------------------------------------


t1$plot_group_distance(distance_pair_stat = TRUE)
t1$plot_group_distance(distance_pair_stat = TRUE, hide_ns = TRUE)
t1$plot_group_distance(distance_pair_stat = TRUE, hide_ns = TRUE, hide_ns_more = c("ns", "*"))
t1$plot_group_distance(distance_pair_stat = TRUE, pair_compare_filter_select = 3)


## ------------------------------------------------
## Method `trans_beta$plot_clustering`
## ------------------------------------------------

t1$plot_clustering(group = "Group", replace_name = c("Saline", "Type"))
```

`trans_classifier` *Create trans_classifier object for machine-learning-based model prediction.*

Description

This class is a wrapper for methods of machine-learning-based classification models, including data pre-processing, feature selection, data split, model training, prediction, confusionMatrix and ROC (Receiver Operator Characteristic) or PR (Precision-Recall) curve.

Author(s): Felipe Mansoldo and Chi Liu

Methods

Public methods:

- trans_classifier$new()
- trans_classifier$cal_preProcess()
- trans_classifier$cal_feature_sel()
- trans_classifier$cal_split()
- trans_classifier$set_trainControl()
- trans_classifier$cal_train()
- trans_classifier$cal_feature_imp()
- trans_classifier$cal_predict()
- trans_classifier$plot_confusionMatrix()
- trans_classifier$cal_ROC()
- trans_classifier$plot_ROC()
- trans_classifier$clone()

Method `new()`**:** Create the trans_classifier object.

Usage:

```
trans_classifier$new(
  dataset = NULL,
  x.predictors = "all",
  y.response = NULL,
  n.cores = 1
)
```

Arguments:

`dataset` the object of `microtable` Class.

`x.predictors` default "all"; character string or data.frame; a character string represents selecting the corresponding data from microtable$taxa_abund; data.frame represents other customized data. See the following available options:

'all' use all the taxa stored in microtable$taxa_abund

'Genus' use Genus level table in microtable$taxa_abund, or other specific taxonomic rank, e.g. 'Phylum'

other input must be a data.frame; It should have the same format with the data.frame in microtable$taxa_abund, i.e. rows are features; cols are samples with same names in sample_table

`y.response` default NULL; the response variable in sample_table.

`n.cores` default 1; the CPU thread used.

Returns: data_feature and data_response in the object.

Examples:

```
\donttest{
data(dataset)
t1 <- trans_classifier$new(
dataset = dataset,
x.predictors = "Genus",
y.response = "Group")
}
```

Method `cal_preProcess()`**:** Pre-process (centering, scaling etc.) of the feature data based on the caret::preProcess function. See https://topepo.github.io/caret/pre-processing.html for more details.

Usage:

```
trans_classifier$cal_preProcess(...)
```

Arguments:

`...` parameters pass to preProcess function of caret package.

Returns: converted data_feature in the object.

Examples:

```
\dontrun{
t1$cal_preProcess(method = c("center", "scale", "nzv"))
}
```

Method `cal_feature_sel()`**:** Perform feature selection. See https://topepo.github.io/caret/feature-selection-overview.html for more details.

Usage:

```
trans_classifier$cal_feature_sel(
  boruta.maxRuns = 300,
  boruta.pValue = 0.01,
  boruta.repetitions = 4,
  ...
)
```

Arguments:

`boruta.maxRuns` default 300; maximal number of importance source runs; passed to the maxRuns parameter in Boruta function of Boruta package.

`boruta.pValue` default 0.01; p value passed to the pValue parameter in Boruta function of Boruta package.

`boruta.repetitions` default 4; repetition runs for the feature selection.

`...` parameters pass to Boruta function of Boruta package.

Returns: optimized data_feature in the object.

Examples:

```
\donttest{
t1$cal_feature_sel(boruta.maxRuns = 300, boruta.pValue = 0.01)
}
```

Method `cal_split()`**:** Split data for training and testing.

Usage:

```
trans_classifier$cal_split(prop.train = 3/4)
```

Arguments:

`prop.train` default 3/4; the ratio of the dataset used for the training.

Returns: data_train and data_test in the object.

Examples:

```
\donttest{
t1$cal_split(prop.train = 3/4)
}
```

Method `set_trainControl()`**:** Control parameters for the following training. See trainControl function of caret package for details.

Usage:

```
trans_classifier$set_trainControl(
  method = "repeatedcv",
  classProbs = TRUE,
  savePredictions = TRUE,
  ...
)
```

Arguments:

`method` default 'repeatedcv'; 'repeatedcv': Repeated k-Fold cross validation; see method parameter in trainControl function of caret package for available options.

`classProbs` default TRUE; should class probabilities be computed for classification models?; see classProbs parameter in caret::trainControl function.

`savePredictions` default TRUE; see savePredictions parameter in caret::trainControl function

`...` parameters pass to trainControl function of caret package.

Returns: trainControl in the object.

Examples:

```
\dontrun{
t1$set_trainControl(method = 'repeatedcv')
}
```

Method `cal_train()`**:** Run the model training.

Usage:

```
trans_classifier$cal_train(
  method = "rf",
  metric = "Accuracy",
  max.mtry = 2,
  max.ntree = 200,
  ...
)
```

Arguments:

`method` default "rf"; "rf": random forest; see method in caret::train function for other options.

`metric` default "Accuracy"; see metric in caret::train function for other options.

`max.mtry` default 2; for method = "rf"; maximum mtry used for the tunegrid to do hyperparameter tuning to optimize the model.

`max.ntree` default 200; for method = "rf"; maximum number of trees used to optimize the model.

`...` parameters pass to train function of caret package.

Returns: res_train in the object.

Examples:

```
\dontrun{
# random forest
t1$cal_train(method = "rf")
# Support Vector Machines with Radial Basis Function Kernel
t1$cal_train(method = "svmRadial", tuneLength = 15)
}
```

Method `cal_feature_imp()`**:** Get feature importance from the training model.

Usage:

```
trans_classifier$cal_feature_imp(...)
```

Arguments:

`...` parameters pass to varImp function of caret package.

Returns: res_feature_imp in the object. One row for each predictor variable. The column(s) are different importance measures.

Examples:

```
\dontrun{
t1$cal_feature_imp()
}
```

Method `cal_predict()`**:** Run the prediction.

Usage:

```
trans_classifier$cal_predict(positive_class = NULL)
```

Arguments:

`positive_class` default NULL; see positive parameter in confusionMatrix function of caret package; If positive_class is NULL, use the first group in data as the positive class automatically.

Returns: res_predict, res_confusion_fit and res_confusion_stats stored in the object.

Examples:

```
\dontrun{
t1$cal_predict()
}
```

Method `plot_confusionMatrix()`**:** Plot the cross-tabulation of observed and predicted classes with associated statistics.

Usage:

```
trans_classifier$plot_confusionMatrix(
  plot_confusion = TRUE,
  plot_statistics = TRUE
)
```

Arguments:

`plot_confusion` default TRUE; whether plot the confusion matrix.

`plot_statistics` default TRUE; whether plot the statistics.

Returns: ggplot object.

Examples:

```
\dontrun{
t1$plot_confusionMatrix()
}
```

Method `cal_ROC()`**:** Get ROC (Receiver Operator Characteristic) curve data and the performance data.

Usage:

```
trans_classifier$cal_ROC(input = "pred")
```

Arguments:

`input` default "pred"; 'pred' or 'train'; 'pred' represents using prediction results; 'train' represents using training results.

Returns: a list res_ROC stored in the object.

Examples:

```
\dontrun{
t1$cal_ROC()
}
```

Method `plot_ROC()`**:** Plot ROC curve.

Usage:

```
trans_classifier$plot_ROC(
  plot_type = c("ROC", "PR")[1],
  plot_group = "all",
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  add_AUC = TRUE,
  plot_method = FALSE,
  ...
)
```

Arguments:

`plot_type` default c("ROC", "PR")[1]; 'ROC' represents ROC (Receiver Operator Characteristic) curve; 'PR' represents PR (Precision-Recall) curve.

`plot_group` default "all"; 'all' represents all the classes in the model; 'add' represents all adding micro-average and macro-average results, see https://scikit-learn.org/stable/auto_examples/model_selection/p other options should be one or more class names, same with the names in Group column of res_ROC$res_roc from cal_ROC function.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors used in the plot.

`add_AUC` default TRUE; whether add AUC in the legend.

`plot_method` default FALSE; If TRUE, show the method in the legend though only one method is found.

`...` parameters pass to geom_path function of ggplot2 package.

Returns: ggplot2 object.

Examples:

```
\dontrun{
t1$plot_ROC(size = 1, alpha = 0.7)
}
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_classifier$new`
## ------------------------------------------------


data(dataset)
t1 <- trans_classifier$new(
dataset = dataset,
x.predictors = "Genus",
y.response = "Group")


## ------------------------------------------------
## Method `trans_classifier$cal_preProcess`
## ------------------------------------------------

## Not run:
t1$cal_preProcess(method = c("center", "scale", "nzv"))

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_feature_sel`
## ------------------------------------------------


t1$cal_feature_sel(boruta.maxRuns = 300, boruta.pValue = 0.01)


## ------------------------------------------------
## Method `trans_classifier$cal_split`
## ------------------------------------------------


t1$cal_split(prop.train = 3/4)


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

```
## Method `trans_classifier$set_trainControl`
## ------------------------------------------------

## Not run:
t1$set_trainControl(method = 'repeatedcv')

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_train`
## ------------------------------------------------

## Not run:
# random forest
t1$cal_train(method = "rf")
# Support Vector Machines with Radial Basis Function Kernel
t1$cal_train(method = "svmRadial", tuneLength = 15)

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_feature_imp`
## ------------------------------------------------

## Not run:
t1$cal_feature_imp()

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_predict`
## ------------------------------------------------

## Not run:
t1$cal_predict()

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$plot_confusionMatrix`
## ------------------------------------------------

## Not run:
t1$plot_confusionMatrix()

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_ROC`
## ------------------------------------------------

## Not run:
t1$cal_ROC()
```

```
## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$plot_ROC`
## ------------------------------------------------

## Not run:
t1$plot_ROC(size = 1, alpha = 0.7)

## End(Not run)
```

`trans_diff` *Create trans_diff object for the differential analysis on the taxonomic abundance.*

Description

This class is a wrapper for a series of differential abundance test and indicator analysis methods, including LEfSe based on the Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>, random forest <doi:10.1016/j.geoderma.2018.09.035>, metastat based on White et al. (2009) <doi:10.1371/journal.pcbi.1000352>, the method in R package metagenomeSeq Paulson et al. (2013) <doi:10.1038/nmeth.2658>, non-parametric Kruskal-Wallis Rank Sum Test, Dunn's Kruskal-Wallis Multiple Comparisons based on the FSA package, Wilcoxon Rank Sum and Signed Rank Tests, t test and anova.

Authors: Chi Liu, Yang Cao, Chenhao Li

Methods

Public methods:

- `trans_diff$new()`
- `trans_diff$plot_diff_abund()`
- `trans_diff$plot_diff_bar()`
- `trans_diff$plot_diff_cladogram()`
- `trans_diff$clone()`

Method `new()`:

Usage:

```
trans_diff$new(
  dataset = NULL,
  method = c("lefse", "rf", "metastat", "mseq", "KW", "KW_dunn", "wilcox", "t.test",
    "anova")[1],
  group = NULL,
  taxa_level = "all",
  filter_thres = 0,
  alpha = 0.05,
  p_adjust_method = "fdr",
```

```
  lefse_subgroup = NULL,
  lefse_min_subsam = 10,
  lefse_norm = 1e+06,
  nresam = 0.6667,
  boots = 30,
  rf_ntree = 1000,
  group_choose_paired = NULL,
  mseq_count = 1,
  ...
)
```

Arguments:

`dataset` the object of `microtable` Class.

`method` default "lefse"; see the following available options:

- **'lefse'** LEfSe method based on Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>
- **'rf'** random forest and non-parametric test method based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>
- **'metastat'** Metastat method for all paired groups based on White et al. (2009) <doi:10.1371/journal.pcbi.1000352>
- **'mseq'** zero-inflated log-normal model-based differential test method from metagenomeSeq package.
- **'KW'** KW: Kruskal-Wallis Rank Sum Test for all groups (>= 2)
- **'KW_dunn'** Dunn's Kruskal-Wallis Multiple Comparisons when group number > 2; see dunnTest function in FSA package
- **'wilcox'** Wilcoxon Rank Sum and Signed Rank Tests for all paired groups
- **'t.test'** Student's t-Test for all paired groups
- **'anova'** Duncan's multiple range test for anova

`group` default NULL; sample group used for the comparision; a colname of microtable$sample_table.

`taxa_level` default "all"; 'all' represents using abundance data at all taxonomic ranks; For testing at a specific rank, provide taxonomic rank name, such as "Genus"; this parameter can be applied when method != "mseq"; 'mseq' method is performed on the feature abudance, i.e. microtable$otu_table.

`filter_thres` default 0; the relative abundance threshold used for method != "metastat" or "mseq".

`alpha` default 0.05; differential significance threshold for method = "lefse" or "rf"; used to select taxa with significance across groups.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of p.adjust function for other available options; NULL can disuse the p value adjustment.

`lefse_subgroup` default NULL; sample sub group used for sub-comparision in lefse; Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>.

`lefse_min_subsam` default 10; sample numbers required in the subgroup test.

`lefse_norm` default 1000000; scale value in lefse.

`nresam` default 0.6667; sample number ratio used in each bootstrap for method = "lefse" or "rf".

`boots` default 30; bootstrap test number for method = "lefse" or "rf".

`rf_ntree` default 1000; see ntree in randomForest function of randomForest package when method = "rf".

group_choose_paired default NULL; a vector used for selecting the required groups for paired testing, only used for method = "metastat" or "mseq".

mseq_count default 1; Filter features to have at least 'counts' counts.; see the count parameter in MRcoefs function of metagenomeSeq package.

... parameters passed to cal_diff function of trans_alpha class when method is one of "KW", "KW_dunn", "wilcox", "t.test" and "anova".

Returns: res_diff and res_abund.
res_abund includes mean abudance of each taxa (Mean), standard deviation (SD), standard error (SE) and sample number (N) in the group (Group).
res_diff is the detailed differential test result, containing:
"Comparison": The groups for the comparision, maybe all groups or paired groups. If this column is not found, all groups used;
"Group": Which group has the maximum median or mean value across the test groups; For non-parametric methods, median value; For t.test, mean value;
"Taxa": which taxa is used in this comparision;
"Method": Test method used in the analysis depending on the method input;
"LDA" or "MeanDecreaseGini": LDA: linear discriminant score in LEfSe; MeanDecreaseGini: mean decreasing gini index in random forest;
"P.unadj" and "P.adj": raw p value; P.adj: adjusted p value;
"qvalue": qvalue for metastat analysis.

Examples:

```
\donttest{
data(dataset)
t1 <- trans_diff$new(dataset = dataset, method = "lefse", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "rf", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "metastat", group = "Group", taxa_level = "Genus")
t1 <- trans_diff$new(dataset = dataset, method = "wilcox", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "KW_dunn", group = "Group", taxa_level = "Phylum")
}
```

Method `plot_diff_abund()`**:** Plotting the abundance of differential taxa.

Usage:

```
trans_diff$plot_diff_abund(
  use_number = 1:20,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  select_group = NULL,
  select_taxa = NULL,
  simplify_names = TRUE,
  keep_prefix = TRUE,
  group_order = NULL,
  barwidth = 0.9,
  use_se = TRUE,
  add_sig = FALSE,
  add_sig_label = "Significance",
  add_sig_label_color = "black",
  add_sig_tip_length = 0.01,
  y_start = 1.01,
```

```
  y_increase = 0.05,
  text_y_size = 10,
  coord_flip = TRUE,
  ...
)
```

Arguments:

`use_number` default 1:20; numeric vector; the taxa numbers (1:n) used in the plot; If the n is larger than the number of total significant taxa, automatically use all the taxa.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette.

`select_group` default NULL; this is used to select the paired groups. This parameter is especially useful when the comparision methods is applied to paired groups; The input select_group must be one of objectres_diffComparison.

`select_taxa` default NULL; character vector to provide taxa names. The taxa names should be same with the names shown in the plot, not the 'Taxa' column names in objectres_diffTaxa.

`simplify_names` default TRUE; whether use the simplified taxonomic name.

`keep_prefix` default TRUE; whether retain the taxonomic prefix.

`group_order` default NULL; a vector to order groups, i.e. reorder the legend and colors in plot; If NULL, the function can first check whether the group column of sample_table is factor. If yes, use the levels in it. If provided, overlook the levels in the group of sample_table.

`barwidth` default 0.9; the bar width in plot.

`use_se` default TRUE; whether use SE in plot, if FALSE, use SD.

`add_sig` default FALSE; whether add the significance label to the plot.

`add_sig_label` default "Significance"; select a colname of object$res_diff for the label text, such as 'P.adj' or 'Significance'.

`add_sig_label_color` default "black"; the color for the label text when add_sig = TRUE.

`add_sig_tip_length` default 0.01; the tip length for the added line when add_sig = TRUE.

`y_start` default 1.01; the y axis position from which to add the label; the default 1.01 means 1.01 * Value; For method != "anova", all the start positions are same, i.e. Value = max(Mean+SD or Mean+SE); For method = "anova"; the stat position is calculated for each point, i.e. Value = Mean+SD or Mean+SE.

`y_increase` default 0.05; the increasing y axia space to add label for paired groups; the default 0.05 means 0.05 * y_start * Value; In addition, this parameter is also used to label the letters of anova result with the fixed (1 + y_increase) * y_start * Value.

`text_y_size` default 10; the size for the y axis text.

`coord_flip` default TRUE; whether flip cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal.

`...` parameters passed to ggsignif::stat_signif when add_sig = TRUE.

Returns: ggplot.

Examples:

```
\donttest{
t1 <- trans_diff$new(dataset = dataset, method = "anova", group = "Group", taxa_level = "Genus")
t1$plot_diff_abund(use_number = 1:10)
t1$plot_diff_abund(use_number = 1:10, add_sig = TRUE)
t1 <- trans_diff$new(dataset = dataset, method = "wilcox", group = "Group")
```

```
t1$plot_diff_abund(use_number = 1:20)
t1$plot_diff_abund(use_number = 1:20, add_sig = TRUE)
t1 <- trans_diff$new(dataset = dataset, method = "lefse", group = "Group")
t1$plot_diff_abund(use_number = 1:20)
t1$plot_diff_abund(use_number = 1:20, add_sig = TRUE)
}
```

Method `plot_diff_bar()`**:** Bar plot for LDA score.

Usage:

```
trans_diff$plot_diff_bar(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  use_number = 1:10,
  threshold = NULL,
  select_group = NULL,
  simplify_names = TRUE,
  keep_prefix = TRUE,
  group_order = NULL,
  axis_text_y = 12,
  plot_vertical = TRUE,
  ...
)
```

Arguments:

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette for different groups.

`use_number` default 1:10; numeric vector; the taxa numbers used in the plot, i.e. 1:n.

`threshold` default NULL; threshold value for selecting taxa, such as 3 for LDA score of LEfSe.

`select_group` default NULL; this is used to select the paired group when multiple comparisions are generated; The input select_group must be one of objectres_diffComparison.

`simplify_names` default TRUE; whether use the simplified taxonomic name.

`keep_prefix` default TRUE; whether retain the taxonomic prefix.

`group_order` default NULL; a vector to order the legend and colors in plot; If NULL, the function can first check whether the group column of sample_table is factor. If yes, use the levels in it. If provided, this parameter can overwrite the levels in the group of sample_table.

`axis_text_y` default 12; the size for the y axis text.

`plot_vertical` default TRUE; whether use vertical bar plot or horizontal.

`...` parameters pass to `geom_bar`

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_diff_bar(use_number = 1:20)
}
```

Method `plot_diff_cladogram()`**:** Plot the cladogram using taxa with significant difference.

Usage:

```
trans_diff$plot_diff_cladogram(
  color = RColorBrewer::brewer.pal(8, "Dark2"),
  use_taxa_num = 200,
  filter_taxa = NULL,
  use_feature_num = NULL,
  group_order = NULL,
  clade_label_level = 4,
  select_show_labels = NULL,
  only_select_show = FALSE,
  sep = "|",
  branch_size = 0.2,
  alpha = 0.2,
  clade_label_size = 2,
  clade_label_size_add = 5,
  clade_label_size_log = exp(1),
  node_size_scale = 1,
  node_size_offset = 1,
  annotation_shape = 22,
  annotation_shape_size = 5
)
```

Arguments:

`color` default RColorBrewer::brewer.pal(8, "Dark2"); color palette used in the plot.

`use_taxa_num` default 200; integer; The taxa number used in the background tree plot; select the taxa according to the mean abundance .

`filter_taxa` default NULL; The mean relative abundance used to filter the taxa with low abundance.

`use_feature_num` default NULL; integer; The feature number used in the plot; select the features according to the LDA score (method = "lefse") or MeanDecreaseGini (method = "rf") from high to low.

`group_order` default NULL; a vector to order the legend and colors in plot; If NULL, the function can first check whether the group column of sample_table is factor. If yes, use the levels in it. If provided, this parameter can overwrite the levels in the group of sample_table.

`clade_label_level` default 4; the taxonomic level for marking the label with letters, root is the largest.

`select_show_labels` default NULL; character vector; The features to show in the plot with full label names, not the letters.

`only_select_show` default FALSE; whether only use the the select features in the parameter select_show_labels.

`sep` default "|"; the seperate character in the taxonomic information.

`branch_size` default 0.2; numberic, size of branch.

`alpha` default 0.2; shading of the color.

`clade_label_size` default 2; basic size for the clade label; please also see clade_label_size_add and clade_label_size_log

`clade_label_size_add` default 5; added basic size for the clade label; see the formula in clade_label_size_log parameter.

`clade_label_size_log` default exp(1); the base of log function for added size of the clade label; the size formula: clade_label_size + log(clade_label_level + clade_label_size_add,

base = clade_label_size_log); so use clade_label_size_log, clade_label_size_add and clade_label_size can totally control the label size for different taxonomic levels.

`node_size_scale` default 1; scale for the node size.

`node_size_offset` default 1; offset for the node size.

`annotation_shape` default 22; shape used in the annotation legend.

`annotation_shape_size` default 5; size used in the annotation legend.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_diff_cladogram(use_taxa_num = 100, use_feature_num = 30, select_show_labels = NULL)
}
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_diff$new`
## ------------------------------------------------


data(dataset)
t1 <- trans_diff$new(dataset = dataset, method = "lefse", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "rf", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "metastat", group = "Group", taxa_level = "Genus")
t1 <- trans_diff$new(dataset = dataset, method = "wilcox", group = "Group")
t1 <- trans_diff$new(dataset = dataset, method = "KW_dunn", group = "Group", taxa_level = "Phylum")


## ------------------------------------------------
## Method `trans_diff$plot_diff_abund`
## ------------------------------------------------


t1 <- trans_diff$new(dataset = dataset, method = "anova", group = "Group", taxa_level = "Genus")
t1$plot_diff_abund(use_number = 1:10)
t1$plot_diff_abund(use_number = 1:10, add_sig = TRUE)
t1 <- trans_diff$new(dataset = dataset, method = "wilcox", group = "Group")
t1$plot_diff_abund(use_number = 1:20)
t1$plot_diff_abund(use_number = 1:20, add_sig = TRUE)
t1 <- trans_diff$new(dataset = dataset, method = "lefse", group = "Group")
t1$plot_diff_abund(use_number = 1:20)
t1$plot_diff_abund(use_number = 1:20, add_sig = TRUE)
```

```
## ------------------------------------------------
## Method `trans_diff$plot_diff_bar`
## ------------------------------------------------


t1$plot_diff_bar(use_number = 1:20)


## ------------------------------------------------
## Method `trans_diff$plot_diff_cladogram`
## ------------------------------------------------


t1$plot_diff_cladogram(use_taxa_num = 100, use_feature_num = 30, select_show_labels = NULL)
```

`trans_env` *Create trans_env object for the analysis of the effects of environmental factors on communities.*

Description

This class is a wrapper for a series of operations associated with environmental measurements, including redundancy analysis, mantel test, correlation analysis and linear fitting based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>.

Methods

Public methods:

- `trans_env$new()`
- `trans_env$cal_diff()`
- `trans_env$cal_autocor()`
- `trans_env$cal_ordination()`
- `trans_env$cal_ordination_envsquare()`
- `trans_env$trans_ordination()`
- `trans_env$plot_ordination()`
- `trans_env$cal_mantel()`
- `trans_env$cal_cor()`
- `trans_env$plot_cor()`
- `trans_env$plot_scatterfit()`
- `trans_env$print()`
- `trans_env$clone()`

Method `new()`:

Usage:

```
trans_env$new(
  dataset = NULL,
  env_cols = NULL,
  add_data = NULL,
  character2numeric = TRUE,
  complete_na = FALSE
)
```

Arguments:

`dataset` the object of `microtable` Class.

`env_cols` default NULL; either numeric vector or character vector to select columns in sample_table of your microtable object. This parameter should be used in the case that all the required environmental data is in sample_table of your microtable object. Otherwise, please use add_data parameter.

`add_data` default NULL; data.frame format; provide the environmental data in the format data.frame; rownames should be sample names. This parameter should be used when the sample_table in your microtable object has no environmental data. Under this circumstance, the env_cols parameter can not be used because no data can be selected.

`character2numeric` default TRUE; whether transform the characters or factors to numeric attributes.

`complete_na` default FALSE; Whether fill the NA (missing value) in the environmental data; If TRUE, the function can run the interpolation with the mice package; to use this parameter, please first install mice package.

Returns: data_env in trans_env object.

Examples:

```
data(dataset)
data(env_data_16S)
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S[, 4:11])
```

Method `cal_diff()`**:** Test the difference of environmental variable across groups.

Usage:

```
trans_env$cal_diff(
  group = NULL,
  method = c("KW", "KW_dunn", "wilcox", "t.test", "anova")[1],
  measure = NULL,
  p_adjust_method = "fdr",
  anova_set = NULL,
  ...
)
```

Arguments:

`group` default NULL; a colname of sample_table used to compare values across groups.

`method` default "KW"; see the following available options:

'KW' KW: Kruskal-Wallis Rank Sum Test for all groups (>= 2)

'KW_dunn' Dunn's Kruskal-Wallis Multiple Comparisons, see dunnTest function in FSA package

'wilcox' Wilcoxon Rank Sum and Signed Rank Tests for all paired groups

't.test' Student's t-Test for all paired groups

'anova' Duncan's multiple range test for anova

`measure` default NULL; a vector; if null, all variables will be calculated.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of p.adjust function for available options.

`anova_set` default NULL; specified group set for anova, such as 'block + N*P*K', see aov.

`...` parameters passed to kruskal.test or wilcox.test function (method = "KW") or dunnTest function of FSA package (method = "KW_dunn") or agricolae::duncan.test (method = "anova").

Returns: res_diff in object. A data.frame generally. A list for anova when anova_set is assigned. In the data frame, 'Group' column means that the group has the maximum median or mean value across the test groups; For non-parametric methods, maximum median value; For t.test, maximum mean value.

Examples:

```
\donttest{
t1$cal_diff(group = "Group", method = "KW")
t1$cal_diff(group = "Group", method = "KW_dunn")
t1$cal_diff(group = "Group", method = "anova")
}
```

Method `cal_autocor()`**:** Calculate the autocorrelations among environmental variables and plot the result.

Usage:

```
trans_env$cal_autocor(
  group = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  alpha = 0.8,
  ...
)
```

Arguments:

`group` default NULL; a colname of sample_table; used to perform calculations for different groups.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); colors palette.

`alpha` default 0.8; the alpha value to add transparency in colors; useful when group is not NULL.

`...` default parameters passed to GGally::ggpairs.

Returns: ggmatrix.

Examples:

```
\donttest{
t1$cal_autocor(method = "GGally")
}
```

Method `cal_ordination()`**:** Redundancy analysis (RDA) and Correspondence Analysis (CCA) based on the vegan package.

Usage:

```
trans_env$cal_ordination(
  method = c("RDA", "dbRDA", "CCA")[1],
  feature_sel = FALSE,
  taxa_level = NULL,
  taxa_filter_thres = NULL,
  use_measure = NULL,
  add_matrix = NULL,
  ...
)
```

Arguments:

`method` default c("RDA", "dbRDA", "CCA")[1]; the ordination method.

`feature_sel` default FALSE; whether perform the feature selection based on forward selection method.

`taxa_level` default NULL; If use RDA or CCA, provide the taxonomic rank, such as "Phylum" or "Genus"; If use otu_table; please input "OTU".

`taxa_filter_thres` default NULL; If want to filter taxa, provide the relative abundance threshold.

`use_measure` default NULL; a name of beta diversity matrix; only useful when parameter method = "dbRDA"; If not provided, use the first beta diversity matrix automatically.

`add_matrix` default NULL; additional distance matrix provided, when the user does not want to use the beta diversity matrix within the dataset; only available when method = "dbRDA".

`...` paremeters pass to dbrda or rda or cca function according to the input of method.

Returns: res_ordination, res_ordination_R2, res_ordination_terms and res_ordination_axis in object.

Examples:

```
\donttest{
t1$cal_ordination(method = "dbRDA", use_measure = "bray")
t1$cal_ordination(method = "RDA", taxa_level = "Genus")
t1$cal_ordination(method = "CCA", taxa_level = "Genus")
}
```

Method `cal_ordination_envsquare()`**:** Fits each environmental vector onto the ordination to obtain the contribution of each variable.

Usage:

```
trans_env$cal_ordination_envsquare(...)
```

Arguments:

`...` the parameters passing to vegan::envfit function.

Returns: res_ordination_envsquare in object.

Examples:

```
\donttest{
t1$cal_ordination_envsquare()
}
```

Method `trans_ordination()`**:** transform ordination result for the following plotting.

Usage:

```
trans_env$trans_ordination(
  show_taxa = 10,
  adjust_arrow_length = FALSE,
  min_perc_env = 0.1,
  max_perc_env = 0.8,
  min_perc_tax = 0.1,
  max_perc_tax = 0.8
)
```

Arguments:

`show_taxa` default 10; taxa number shown in the plot.

`adjust_arrow_length` default FALSE; whether adjust the arrow length to be clearer.

`min_perc_env` default 0.1; used for scaling up the minimum of env arrow; multiply by the maximum distance between samples and origin.

`max_perc_env` default 0.8; used for scaling up the maximum of env arrow; multiply by the maximum distance between samples and origin.

`min_perc_tax` default 0.1; used for scaling up the minimum of tax arrow; multiply by the maximum distance between samples and origin.

`max_perc_tax` default 0.8; used for scaling up the maximum of tax arrow; multiply by the maximum distance between samples and origin.

Returns: res_ordination_trans in object.

Examples:

```
\donttest{
t1$trans_ordination(adjust_arrow_length = TRUE, min_perc_env = 0.1, max_perc_env = 1)
}
```

Method `plot_ordination()`**:** plot ordination result.

Usage:

```
trans_env$plot_ordination(
  plot_color = NULL,
  plot_shape = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  env_text_color = "black",
  env_arrow_color = "grey30",
  taxa_text_color = "firebrick1",
  taxa_arrow_color = "firebrick1",
  env_text_size = 3.7,
  taxa_text_size = 3,
  taxa_text_italic = TRUE,
  plot_type = "point",
  point_size = 3,
  point_alpha = 0.8,
  centroid_segment_alpha = 0.6,
  centroid_segment_size = 1,
  centroid_segment_linetype = 3,
```

```
  ellipse_chull_fill = TRUE,
  ellipse_chull_alpha = 0.1,
  ellipse_level = 0.9,
  ellipse_type = "t",
  add_sample_label = NULL,
  env_nudge_x = NULL,
  env_nudge_y = NULL,
  taxa_nudge_x = NULL,
  taxa_nudge_y = NULL,
  ...
)
```

Arguments:

`plot_color` default NULL; a colname of sample_table to assign colors to different groups in plot.

`plot_shape` default NULL; a colname of sample_table to assign shapes to different groups in plot.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); color pallete for different groups.

`shape_values` default c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14); a vector for point shape types of groups, see ggplot2 tutorial.

`env_text_color` default "black"; environmental variable text color.

`env_arrow_color` default "grey30"; environmental variable arrow color.

`taxa_text_color` default "firebrick1"; taxa text color.

`taxa_arrow_color` default "firebrick1"; taxa arrow color.

`env_text_size` default 3.7; environmental variable text size.

`taxa_text_size` default 3; taxa text size.

`taxa_text_italic` default TRUE; "italic"; whether use "italic" style for the taxa text in the plot.

`plot_type` default "point"; one or more elements of "point", "ellipse", "chull" and "centroid".

- **'point'** add point
- **'ellipse'** add confidence ellipse for points of each group
- **'chull'** add convex hull for points of each group
- **'centroid'** add centroid line of each group

`point_size` default 3; point size in plot when "point" is in plot_type.

`point_alpha` default .8; point transparency in plot when "point" is in plot_type.

`centroid_segment_alpha` default 0.6; segment transparency in plot when "centroid" is in plot_type.

`centroid_segment_size` default 1; segment size in plot when "centroid" is in plot_type.

`centroid_segment_linetype` default 3; an integer; the line type related with centroid in plot when "centroid" is in plot_type.

`ellipse_chull_fill` default TRUE; whether fill colors to the area of ellipse or chull.

`ellipse_chull_alpha` default 0.1; color transparency in the ellipse or convex hull depending on whether "ellipse" or "centroid" is in plot_type.

`ellipse_level` default .9; confidence level of ellipse when "ellipse" is in plot_type.

`ellipse_type` default "t"; ellipse type when "ellipse" is in plot_type; see type in `stat_ellipse`.

add_sample_label default NULL; the column name in sample table, if provided, show the point name in plot.

env_nudge_x default NULL; numeric vector to adjust the env text x axis position; passed to nudge_x parameter of geom_text_repel function of ggrepel package; default NULL represents automatic adjustment; the length must be same with the row number of object$res_ordination_trans$df_arrows. For example, if there are 5 env variables, env_nudge_x should be something like c(0.1, 0, -0.2, 0, 0). Note that this parameter and env_nudge_y is generally used when the automatic text adjustment is not very well.

env_nudge_y default NULL; numeric vector to adjust the env text y axis position; passed to nudge_y parameter of ggrepel::geom_text_repel function; default NULL represents automatic adjustment; the length must be same with the row number of object$res_ordination_trans$df_arrows. For example, if there are 5 env variables, env_nudge_y should be something like c(0.1, 0, -0.2, 0, 0).

taxa_nudge_x default NULL; numeric vector to adjust the taxa text x axis position; passed to nudge_x parameter of ggrepel::geom_text_repel function; default NULL represents automatic adjustment; the length must be same with the row number of object$res_ordination_trans$df_arrows_spe. For example, if 3 taxa are shown, taxa_nudge_x should be something like c(0.3, -0.2, 0).

taxa_nudge_y default NULL; numeric vector to adjust the taxa text y axis position; passed to nudge_y parameter of ggrepel::geom_text_repel function; default NULL represents automatic adjustment; the length must be same with the row number of object$res_ordination_trans$df_arrows_spe. For example, if 3 taxa are shown, taxa_nudge_y should be something like c(-0.2, 0, 0.4).

... paremeters pass to geom_point for controlling sample points.

Returns: ggplot object.

Examples:

```
\donttest{
t1$cal_ordination(method = "RDA")
t1$trans_ordination(adjust_arrow_length = TRUE, max_perc_env = 1.5)
t1$plot_ordination(plot_color = "Group")
t1$plot_ordination(plot_color = "Group", plot_shape = "Group", plot_type = c("point", "ellipse"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "chull"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "centroid"),
  centroid_segment_linetype = 1)
t1$plot_ordination(plot_color = "Group", env_nudge_x = c(0.4, 0, 0, 0, 0, -0.2, 0, 0),
  env_nudge_y = c(0.6, 0, 0.2, 0.5, 0, 0.1, 0, 0.2))
}
```

Method `cal_mantel()`**:** Mantel test between beta diversity matrix and environmental data.

Usage:

```
trans_env$cal_mantel(
  select_env_data = NULL,
  partial_mantel = FALSE,
  add_matrix = NULL,
  use_measure = NULL,
  method = "pearson",
  p_adjust_method = "fdr",
  ...
)
```

Arguments:

`select_env_data` default NULL; numeric or character vector to select columns in data_env; if not provided, automatically select the columns with numeric attributes.

`partial_mantel` default FALSE; whether use partial mantel test; If TRUE, use other measurements as the zdis.

`add_matrix` default NULL; additional distance matrix provided, if you donot want to use the beta diversity matrix in the dataset.

`use_measure` default NULL; name of beta diversity matrix. If necessary and not provided, use the first beta diversity matrix.

`method` default "pearson"; one of "pearson", "spearman" and "kendall"; correlation method; see method parameter in mantel function of vegan package.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of p.adjust function for available options.

`...` paremeters pass to `mantel` of vegan package.

Returns: res_mantel in object.

Examples:

```
\donttest{
t1$cal_mantel(use_measure = "bray")
t1$cal_mantel(partial_mantel = TRUE, use_measure = "bray")
}
```

Method `cal_cor()`**:** Calculating the correlations between taxa abundance and environmental variables. Actually, it can also be used for calculating other correlation between any two variables from two tables.

Usage:

```
trans_env$cal_cor(
  use_data = c("Genus", "all", "other")[1],
  select_env_data = NULL,
  cor_method = c("pearson", "spearman", "kendall")[1],
  p_adjust_method = "fdr",
  p_adjust_type = c("Type", "Taxa", "Env")[3],
  add_abund_table = NULL,
  by_group = NULL,
  use_taxa_num = NULL,
  other_taxa = NULL,
  group_use = NULL,
  group_select = NULL,
  taxa_name_full = TRUE
)
```

Arguments:

`use_data` default "Genus"; "Genus", "all" or "other"; "Genus" or other taxonomic name: use genus or other taxonomic abundance table in taxa_abund; "all": use all merged taxa abundance table; "other": provide additional taxa name with other_taxa parameter which is necessary.

`select_env_data` default NULL; numeric or character vector to select columns in data_env; if not provided, automatically select the columns with numeric attributes.

`cor_method` default "pearson"; "pearson", "spearman" or "kendall"; correlation method.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of p.adjust function for available options.

`p_adjust_type` default "Env"; "Type", "Taxa" or "Env"; p.adjust type; Env: environmental data; Taxa: taxa data; Type: group used.

`add_abund_table` default NULL; additional data table to be used. Samples must be rows.

`by_group` default NULL; one column name or number in sample_table; calculate correlations for different groups separately.

`use_taxa_num` default NULL; integer; a number used to select high abundant taxa; only useful when use_data parameter is a taxonomic level, e.g., "Genus".

`other_taxa` default NULL; character vector containing a series of taxa names; used when use_data = "other"; the provided names should be standard full names used to select taxa from all the tables in taxa_abund list of the microtable object; please see the example.

`group_use` default NULL; numeric or character vector to select one column in sample_table for selecting samples; together with group_select.

`group_select` default NULL; the group name used; remain samples within the group.

`taxa_name_full` default TRUE; Whether use the complete taxonomic name of taxa.

Returns: res_cor in object.

Examples:

```
\donttest{
t2 <- trans_diff$new(dataset = dataset, method = "rf", group = "Group", rf_taxa_level = "Genus")
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S[, 4:11])
t1$cal_cor(use_data = "other", p_adjust_method = "fdr", other_taxa = t2$res_diff$Taxa[1:40])
}
```

Method `plot_cor()`**:** Plot correlation heatmap.

Usage:

```
trans_env$plot_cor(
  color_vector = c("#053061", "white", "#A50026"),
  color_palette = NULL,
  pheatmap = FALSE,
  filter_feature = NULL,
  ylab_type_italic = FALSE,
  keep_full_name = FALSE,
  keep_prefix = TRUE,
  text_y_order = NULL,
  text_x_order = NULL,
  font_family = NULL,
  cluster_ggplot = "none",
  cluster_height_rows = 0.2,
  cluster_height_cols = 0.2,
  text_y_position = "right",
  mylabels_x = NULL,
  ...
)
```

Arguments:

color_vector default c("#053061", "white", "#A50026"); colors with only three values representing low, middle and high value.

color_palette default NULL; a customized palette with more color values; if provided, use it instead of color_vector.

pheatmap default FALSE; whether use pheatmap package to plot the heatmap.

filter_feature default NULL; character vector; used to filter features that only have significance labels in the filter_feature vector. For example, filter_feature = "" can be used to filter features that only have "", no any "*".

ylab_type_italic default FALSE; whether use italic type for y lab text.

keep_full_name default FALSE; whether use the complete taxonomic name.

keep_prefix default TRUE; whether retain the taxonomic prefix.

text_y_order default NULL; character vector; provide customized text order for y axis; shown in the plot from the top down.

text_x_order default NULL; character vector; provide customized text order for x axis.

font_family default NULL; font family used in ggplot2; only available when pheatmap = FALSE.

cluster_ggplot default "none"; add clustering dendrogram for ggplot2 based heatmap; available options: "none", "row", "col" or "both". "none": no any clustering used; "row": add clustering for rows; "col": add clustering for columns; "both": add clustering for both rows and columns. Only available when pheatmap = FALSE.

cluster_height_rows default 0.2, the dendrogram plot height for rows; available when cluster_ggplot != "none".

cluster_height_cols default 0.2, the dendrogram plot height for columns; available cluster_ggplot != "none".

text_y_position default "right"; "left" or "right"; the y axis text position; ggplot2 based heatmap.

mylabels_x default NULL; provide x axis text labels additionally; only available when pheatmap = TRUE.

... paremeters pass to ggplot2::geom_tile or pheatmap, depending on the pheatmap = FALSE or TRUE.

Returns: plot.

Examples:

```
\donttest{
t1$plot_cor(pheatmap = FALSE)
}
```

Method `plot_scatterfit()`**:** Scatter plot and add fitted line. The most important thing is to make sure that the input x and y have correponding sample orders. If one of x and y is a matrix, the other will be also transformed to matrix with Euclidean distance. Then, both of them are transformed to be vectors. If x or y is a vector with a single value, x or y will be assigned according to the column selection of the data_env inside.

Usage:

```
trans_env$plot_scatterfit(
  x = NULL,
  y = NULL,
```

```
  group = NULL,
  group_order = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = NULL,
  type = c("cor", "lm")[1],
  cor_method = "pearson",
  label_sep = ";",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  x_axis_title = "",
  y_axis_title = "",
  point_size = 5,
  point_alpha = 0.6,
  line_size = 0.8,
  line_alpha = 1,
  line_color = "black",
  line_se = TRUE,
  line_se_color = "grey70",
  pvalue_trim = 4,
  cor_coef_trim = 3,
  lm_fir_trim = 2,
  lm_sec_trim = 2,
  lm_squ_trim = 2,
  ...
)
```

Arguments:

`x` default NULL; a single numeric or character value or a vector or a distance matrix used for the x axis. If x is a single value, it will be used to select the column of data_env inside. If x is a distance matrix, it will be transformed to be a vector.

`y` default NULL; a single numeric or character value or a vector or a distance matrix used for the y axis. If y is a single value, it will be used to select the column of data_env inside. If y is a distance matrix, it will be transformed to be a vector.

`group` default NULL; a character vector; if length is 1, must be a colname of dataset$sample_table; Otherwise, group should be a vector with same length of x/y (for vector) or ncol of x/y (for matrix).

`group_order` default NULL; a vector to order groups, i.e. reorder the legend and colors in plot when group is not NULL; If group_order is NULL and group is provided, the function can first check whether the group column of dataset$sample_table is factor. If provided, overlook the levels in the group of dataset$sample_table.

`color_values` default RColorBrewer::brewer.pal(8, "Dark2"); color pallete for different groups.

`shape_values` default NULL; a numeric vector for point shape types of groups when group is not NULL, see ggplot2 tutorial.

`type` default c("cor", "lm")[1]; "cor": correlation; "lm" for regression.

`cor_method` default "pearson"; one of "pearson", "kendall" and "spearman"; correlation method.

`label_sep` default ";"; the separator string between different label parts.

`label.x.npc` default "left"; can be numeric or character vector of the same length as the number of groups and/or panels. If too short they will be recycled.

numeric value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates"

character allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

`label.y.npc` default "top"; same usage with label.x.npc; see also label.y.npc parameter of stat_cor of ggpubr package.

`label.x` default NULL; x axis absolute position for adding the statictic label.

`label.y` default NULL; x axis absolute position for adding the statictic label.

`x_axis_title` default ""; the title of x axis.

`y_axis_title` default ""; the title of y axis.

`point_size` default 5; point size value.

`point_alpha` default 0.6; alpha value for the point color transparency.

`line_size` default 0.8; line size value.

`line_alpha` default 1; alpha value for the line color transparency.

`line_color` default "black"; fitted line color only useful when group = NULL.

`line_se` default TRUE; Whether show the confidence interval for the fitting.

`line_se_color` default "grey70"; the color to fill the confidence interval when line_se = TRUE.

`pvalue_trim` default 4; trim the decimal places of p value.

`cor_coef_trim` default 3; trim the decimal places of correlation coefficient.

`lm_fir_trim` default 2; trim the decimal places of regression first coefficient.

`lm_sec_trim` default 2; trim the decimal places of regression second coefficient.

`lm_squ_trim` default 2; trim the decimal places of regression R square.

`...` other arguments to pass to geom_text or geom_label.

Returns: plot.

Examples:

```
\donttest{
t1$plot_scatterfit(x = 1, y = 2, type = "cor")
t1$plot_scatterfit(x = 1, y = 2, type = "lm", point_alpha = .3)
t1$plot_scatterfit(x = "pH", y = "TOC", type = "lm", group = "Group", line_se = FALSE)
t1$plot_scatterfit(x =
 dataset$beta_diversity$bray[rownames(t1$data_env), rownames(t1$data_env)], y = "pH")
}
```

Method `print()`**:** Print the trans_env object.

Usage:

```
trans_env$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_env$new`
## ------------------------------------------------

data(dataset)
data(env_data_16S)
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S[, 4:11])

## ------------------------------------------------
## Method `trans_env$cal_diff`
## ------------------------------------------------


t1$cal_diff(group = "Group", method = "KW")
t1$cal_diff(group = "Group", method = "KW_dunn")
t1$cal_diff(group = "Group", method = "anova")


## ------------------------------------------------
## Method `trans_env$cal_autocor`
## ------------------------------------------------


t1$cal_autocor(method = "GGally")


## ------------------------------------------------
## Method `trans_env$cal_ordination`
## ------------------------------------------------


t1$cal_ordination(method = "dbRDA", use_measure = "bray")
t1$cal_ordination(method = "RDA", taxa_level = "Genus")
t1$cal_ordination(method = "CCA", taxa_level = "Genus")


## ------------------------------------------------
## Method `trans_env$cal_ordination_envsquare`
## ------------------------------------------------


t1$cal_ordination_envsquare()


## ------------------------------------------------
## Method `trans_env$trans_ordination`
## ------------------------------------------------


t1$trans_ordination(adjust_arrow_length = TRUE, min_perc_env = 0.1, max_perc_env = 1)
```

```
## ------------------------------------------------
## Method `trans_env$plot_ordination`
## ------------------------------------------------


t1$cal_ordination(method = "RDA")
t1$trans_ordination(adjust_arrow_length = TRUE, max_perc_env = 1.5)
t1$plot_ordination(plot_color = "Group")
t1$plot_ordination(plot_color = "Group", plot_shape = "Group", plot_type = c("point", "ellipse"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "chull"))
t1$plot_ordination(plot_color = "Group", plot_type = c("point", "centroid"),
  centroid_segment_linetype = 1)
t1$plot_ordination(plot_color = "Group", env_nudge_x = c(0.4, 0, 0, 0, 0, -0.2, 0, 0),
  env_nudge_y = c(0.6, 0, 0.2, 0.5, 0, 0.1, 0, 0.2))


## ------------------------------------------------
## Method `trans_env$cal_mantel`
## ------------------------------------------------


t1$cal_mantel(use_measure = "bray")
t1$cal_mantel(partial_mantel = TRUE, use_measure = "bray")


## ------------------------------------------------
## Method `trans_env$cal_cor`
## ------------------------------------------------


t2 <- trans_diff$new(dataset = dataset, method = "rf", group = "Group", rf_taxa_level = "Genus")
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S[, 4:11])
t1$cal_cor(use_data = "other", p_adjust_method = "fdr", other_taxa = t2$res_diff$Taxa[1:40])


## ------------------------------------------------
## Method `trans_env$plot_cor`
## ------------------------------------------------


t1$plot_cor(pheatmap = FALSE)


## ------------------------------------------------
## Method `trans_env$plot_scatterfit`
## ------------------------------------------------


t1$plot_scatterfit(x = 1, y = 2, type = "cor")
t1$plot_scatterfit(x = 1, y = 2, type = "lm", point_alpha = .3)
t1$plot_scatterfit(x = "pH", y = "TOC", type = "lm", group = "Group", line_se = FALSE)
```

```
t1$plot_scatterfit(x =
 dataset$beta_diversity$bray[rownames(t1$data_env), rownames(t1$data_env)], y = "pH")
```

`trans_func` *Create trans_func object for functional analysis.*

Description

This class is a wrapper for a series of functional analysis on species and communities, including the prokaryotes function identification based on Louca et al. (2016) <doi:10.1126/science.aaf4507> and Lim et al. (2020) <10.1038/s41597-020-0516-5>, or fungi function identification based on Nguyen et al. (2016) <10.1016/j.funeco.2015.06.006> and Polme et al. (2020) <doi:10.1007/s13225-020-00466-2>; functional redundancy calculation and metabolic pathway abundance prediction Abhauer et al. (2015) <10.1093/bioinformatics/btv287>.

Active bindings

`func_group_list` store and show the function group list

Methods

Public methods:

- `trans_func$new()`
- `trans_func$cal_spe_func()`
- `trans_func$cal_spe_func_perc()`
- `trans_func$show_prok_func()`
- `trans_func$plot_spe_func_perc()`
- `trans_func$cal_tax4fun()`
- `trans_func$cal_tax4fun2()`
- `trans_func$cal_tax4fun2_FRI()`
- `trans_func$print()`
- `trans_func$clone()`

Method `new()`**:** Create the trans_func object. This function can identify the data type for Prokaryotes or Fungi automatically.

Usage:

```
trans_func$new(dataset = NULL)
```

Arguments:

`dataset` the object of `microtable` Class.

Returns: for_what : "prok" or "fungi" or NA, "prok" represent prokaryotes. "fungi" represent fungi. NA stand for not identified according to the Kingdom information, at this time, if you want to use the functions to identify species traits, you need provide "prok" or "fungi" manually, e.g. dataset$for_what <- "prok".

Examples:

```
data(dataset)
t1 <- trans_func$new(dataset = dataset)
```

Method `cal_spe_func()`**:** Confirm traits of each feature by matching the taxonomic assignments to the functional database.

Usage:

```
trans_func$cal_spe_func(
  prok_database = c("FAPROTAX", "NJC19")[1],
  fungi_database = c("FUNGuild", "FungalTraits")[1]
)
```

Arguments:

`prok_database` default "FAPROTAX"; "FAPROTAX" or "NJC19"; select a prokaryotic trait database; see the details:

'FAPROTAX' FAPROTAX v1.2.4 Reference: Louca et al. (2016). Decoupling function and taxonomy in the global ocean microbiome. Science, 353(6305), 1272. <doi:10.1126/science.aaf4507>

'NJC19' NJC19: Lim et al. (2020). Large-scale metabolic interaction network of the mouse and human gut microbiota. Scientific Data, 7(1). <10.1038/s41597-020-0516-5>

`fungi_database` default "FUNGuild"; "FUNGuild" or "FungalTraits"; select a fungal trait database; see the details:

'FUNGuild' Nguyen et al. (2016) FUNGuild: An open annotation tool for parsing fungal community datasets by ecological guild. Fungal Ecology, 20(1), 241-248, <doi:10.1016/j.funeco.2015.06.006>

'FungalTraits' Polme et al. FungalTraits: a user-friendly traits database of fungi and fungus-like stramenopiles. Fungal Diversity 105, 1-16 (2020). <doi:10.1007/s13225-020-00466-2>

Returns: res_spe_func stored in object.

Examples:

```
\donttest{
t1$cal_spe_func(prok_database = "FAPROTAX")
t1$cal_spe_func(fungi_database = "FungalTraits")
}
```

Method `cal_spe_func_perc()`**:** Calculating the percentages of species with specific trait in communities. The percentages of the taxa with specific trait can reflect the potential of the corresponding function in the community. So this method is a simple calculation of functional redundancy without the consideration of phylogenetic distance among taxa.

Usage:

```
trans_func$cal_spe_func_perc(abundance_weighted = FALSE)
```

Arguments:

`abundance_weighted` default FALSE; whether use abundance of taxa. If FALSE, calculate the functional population percentage. If TRUE, calculate the functional individual percentage.

Returns: res_spe_func_perc stored in the object.

Examples:

```
\donttest{
t1$cal_spe_func_perc(abundance_weighted = TRUE)
}
```

Method `show_prok_func()`**:** Show the annotation information for a function of prokaryotes from FAPROTAX database.

Usage:

```
trans_func$show_prok_func(use_func = NULL)
```

Arguments:

`use_func` default NULL; the function name.

Returns: None.

Examples:

```
\donttest{
t1$show_prok_func(use_func = "methanotrophy")
}
```

Method `plot_spe_func_perc()`**:** Plot the percentages of species with specific trait in communities or network modules.

Usage:

```
trans_func$plot_spe_func_perc(
  filter_func = NULL,
  use_group_list = TRUE,
  add_facet = TRUE,
  select_samples = NULL,
  color_gradient_low = "#00008B",
  color_gradient_high = "#9E0142"
)
```

Arguments:

`filter_func` default NULL; a vector of function names used to show in the plot.

`use_group_list` default TRUE; If TRUE, use default group list; If user want to use personalized group list, please first set trans_func$func_group_list object with a list of group names and functions.

`add_facet` default TRUE; whether use group names as the facets in the plot, see trans_func$func_group_list object.

`select_samples` default NULL; character vector; select partial samples to show.

`color_gradient_low` default "#00008B"; the color used as the low end in the color gradient.

`color_gradient_high` default "#9E0142"; the color used as the high end in the color gradient.

Returns: ggplot2.

Examples:

```
\donttest{
t1$plot_spe_func_perc(use_group_list = TRUE)
}
```

Method `cal_tax4fun()`**:** Predict functional potential of communities using tax4fun. please cite: Tax4Fun: Predicting functional profiles from metagenomic 16S rRNA data. Bioinformatics, 31(17), 2882-2884, <doi:10.1093/bioinformatics/btv287>. Note that this function requires a standard prefix in taxonomic table with double underlines (e.g. g__) .

Usage:

```
trans_func$cal_tax4fun(keep_tem = FALSE, folderReferenceData = NULL)
```

Arguments:

`keep_tem` default FALSE; whether keep the intermediate file, that is, the feature table in local place.

`folderReferenceData` default NULL; the folder, see http://tax4fun.gobics.de/ and Tax4Fun function in Tax4Fun package.

Returns: tax4fun_KO and tax4fun_path in object.

Method `cal_tax4fun2()`**:** Predict functional potential of communities with Tax4Fun2 method. The function was adapted from the raw Tax4Fun2 package to make it compatible with the microtable object. Pleas cite: Tax4Fun2: prediction of habitat-specific functional profiles and functional redundancy based on 16S rRNA gene sequences. Environmental Microbiome 15, 11 (2020). <doi:10.1186/s40793-020-00358-7>

Usage:

```
trans_func$cal_tax4fun2(
  blast_tool_path = NULL,
  path_to_reference_data = "Tax4Fun2_ReferenceData_v2",
  path_to_temp_folder = NULL,
  database_mode = "Ref99NR",
  normalize_by_copy_number = T,
  min_identity_to_reference = 97,
  use_uproc = T,
  num_threads = 1,
  normalize_pathways = F
)
```

Arguments:

`blast_tool_path` default NULL; the folder path, e.g., ncbi-blast-2.5.0+/bin ; blast tools folder downloaded from "ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+" ; e.g., ncbi-blast-2.5.0+-x64-win64.tar.gz for windows system; if blast_tool_path is NULL, search the tools in the environmental path variable.

`path_to_reference_data` default "Tax4Fun2_ReferenceData_v2"; the path that points to files used in the prediction; The directory must contain the Ref99NR or Ref100NR folder; download Ref99NR.zip from https://cloudstor.aarnet.edu.au/plus/s/DkoZIyZpMNbrzSw/download or Ref100NR.zip from https://cloudstor.aarnet.edu.au/plus/s/jIByczak9ZAFUB4/download.

`path_to_temp_folder` default NULL; The temporary folder to store the logfile, intermediate file and result files; if NULL, use the default temporary in the computer system.

`database_mode` default 'Ref99NR'; "Ref99NR" or "Ref100NR"; Ref99NR: 99% clustering reference database; Ref100NR: no clustering.

`normalize_by_copy_number` default TRUE; whether normalize the result by the 16S rRNA copy number in the genomes.

`min_identity_to_reference` default 97; the idenity threshold used for finding the nearest species.

`use_uproc` default TRUE; whether use UProC to functionally anotate the genomes in the reference data.

`num_threads` default 1; the threads used in the blastn.

`normalize_pathways` default FALSE; Different to Tax4Fun, when converting from KEGG functions to KEGG pathways, Tax4Fun2 does not equally split KO gene abundances between pathways a functions is affiliated to. The full predicted abundance is affiliated to each pathway. Use TRUE to split the abundances (default is FALSE).

Returns: res_tax4fun2_KO and res_tax4fun2_pathway in object.

Examples:

```
\dontrun{
t1$cal_tax4fun2(blast_tool_path = "ncbi-blast-2.5.0+/bin",
    path_to_reference_data = "Tax4Fun2_ReferenceData_v2")
}
```

Method `cal_tax4fun2_FRI()`**:** Calculate (multi-) functional redundancy index (FRI) of prokaryotic community with Tax4Fun2 method. This function is used to calculating aFRI and rFRI use the intermediate files generated by the function cal_tax4fun2(). please also cite: Tax4Fun2: prediction of habitat-specific functional profiles and functional redundancy based on 16S rRNA gene sequences. Environmental Microbiome 15, 11 (2020). <doi:10.1186/s40793-020-00358-7>

Usage:

```
trans_func$cal_tax4fun2_FRI()
```

Returns: res_tax4fun2_aFRI and res_tax4fun2_rFRI in object.

Examples:

```
\dontrun{
t1$cal_tax4fun2_FRI()
}
```

Method `print()`**:** Print the trans_func object.

Usage:

```
trans_func$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_func$new`
## ------------------------------------------------
```

```
data(dataset)
t1 <- trans_func$new(dataset = dataset)

## ------------------------------------------------
## Method `trans_func$cal_spe_func`
## ------------------------------------------------


t1$cal_spe_func(prok_database = "FAPROTAX")
t1$cal_spe_func(fungi_database = "FungalTraits")


## ------------------------------------------------
## Method `trans_func$cal_spe_func_perc`
## ------------------------------------------------


t1$cal_spe_func_perc(abundance_weighted = TRUE)


## ------------------------------------------------
## Method `trans_func$show_prok_func`
## ------------------------------------------------


t1$show_prok_func(use_func = "methanotrophy")


## ------------------------------------------------
## Method `trans_func$plot_spe_func_perc`
## ------------------------------------------------


t1$plot_spe_func_perc(use_group_list = TRUE)


## ------------------------------------------------
## Method `trans_func$cal_tax4fun2`
## ------------------------------------------------

## Not run:
t1$cal_tax4fun2(blast_tool_path = "ncbi-blast-2.5.0+/bin",
    path_to_reference_data = "Tax4Fun2_ReferenceData_v2")

## End(Not run)

## ------------------------------------------------
## Method `trans_func$cal_tax4fun2_FRI`
## ------------------------------------------------

## Not run:
t1$cal_tax4fun2_FRI()
```

```
## End(Not run)
```

`trans_network` *Create trans_network object for co-occurrence network analysis.*

Description

This class is a wrapper for a series of network analysis methods, including the network construction approaches, network attributes analysis, eigengene analysis, network subsetting, node and edge properties extraction, network plotting, and other network operations.

Methods

Public methods:

- `trans_network$new()`
- `trans_network$cal_network()`
- `trans_network$cal_module()`
- `trans_network$save_network()`
- `trans_network$cal_network_attr()`
- `trans_network$cal_node_type()`
- `trans_network$get_node_table()`
- `trans_network$get_edge_table()`
- `trans_network$get_adjacency_matrix()`
- `trans_network$plot_network()`
- `trans_network$cal_eigen()`
- `trans_network$plot_taxa_roles()`
- `trans_network$subset_network()`
- `trans_network$cal_powerlaw()`
- `trans_network$trans_comm()`
- `trans_network$print()`
- `trans_network$clone()`

Method `new()`**:** This function is used to create the trans_network object, store the important intermediate data and calculate correlations if cal_cor parameter is selected.

Usage:

```
trans_network$new(
  dataset = NULL,
  cor_method = c("pearson", "spearman", "kendall")[1],
  cal_cor = c("base", "WGCNA", "SparCC", NA)[1],
  taxa_level = "OTU",
  filter_thres = 0,
  nThreads = 1,
  SparCC_simu_num = 100,
```

```
  env_cols = NULL,
  add_data = NULL
)
```

Arguments:

`dataset` the object of `microtable` Class.

`cor_method` default "pearson"; "pearson", "spearman" or "kendall"; correlation algorithm, only use for correlation-based network.

`cal_cor` default "base"; "base", "WGCNA", "SparCC" or NA; correlation method; NA represents no correlation calculation, used for non-correlation based network, such as SpiecEasi and FlashWeave methods.

`taxa_level` default "OTU"; taxonomic rank; 'OTU' represents using feature table directly; other available options should be one of the colnames of microtable$tax_table.

`filter_thres` default 0; the relative abundance threshold.

`nThreads` default 1; the thread number used for "WGCNA" and SparCC.

`SparCC_simu_num` default 100; SparCC simulation number for bootstrap.

`env_cols` default NULL; numeric or character vector to select the column names of environmental data in dataset$sample_table; the environmental data can be used in the correlation network (as the nodes) or FlashWeave network.

`add_data` default NULL; provide environmental table additionally instead of env_cols parameter; rownames must be sample names.

Returns: res_cor_p list; include the correlation matrix and p value matrix.

Examples:

```
\donttest{
data(dataset)
# for correlation network
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
# for other network
t1 <- trans_network$new(dataset = dataset, cal_cor = NA)
}
```

Method `cal_network()`**:** Calculate network based on the correlation method or SpiecEasi package or julia FlashWeave package or beemStatic package.

Usage:

```
trans_network$cal_network(
  network_method = c("COR", "SpiecEasi", "FlashWeave", "beemStatic")[1],
  COR_p_thres = 0.01,
  COR_p_adjust = "fdr",
  COR_weight = TRUE,
  COR_cut = 0.6,
  COR_optimization = FALSE,
  COR_optimization_low_high = c(0.4, 0.8),
  SpiecEasi_method = "mb",
  FlashWeave_tempdir = NULL,
  FlashWeave_meta_data = FALSE,
```

```
  FlashWeave_other_para = "alpha=0.01,sensitive=true,heterogeneous=true",
  beemStatic_t_strength = 0.001,
  beemStatic_t_stab = 0.8,
  add_taxa_name = "Phylum",
  usename_rawtaxa_when_taxalevel_notOTU = FALSE,
  ...
)
```

Arguments:

`network_method` default "COR"; "COR", "SpiecEasi", "FlashWeave" or "beemStatic"; The option details:

- **'COR'** correlation-based network; use the correlation and p value matrixes in object$res_cor_p returned from trans_network$new; See Deng et al. (2012) <doi:10.1186/1471-2105-13-113> for other details
- **'SpiecEasi'** SpiecEasi network; relies on algorithms for sparse neighborhood and inverse covariance selection; belong to the category of conditional dependence and graphical models; see https://github.com/zdk123/SpiecEasi for installing the R package; see Kurtz et al. (2015) <doi:10.1371/journal.pcbi.1004226> for the algorithm details
- **'FlashWeave'** FlashWeave network; Local-to-global learning framework; belong to the category of conditional dependence and graphical models; good performance on heterogenous datasets to find direct associations among taxa; see https://github.com/meringlab/FlashWeave.jl for installing julia language and FlashWeave package; julia must be in the computer system env path, otherwise the program can not find julia; see Tackmann et al. (2019) <doi:10.1016/j.cels.2019.08.002> for the algorithm details
- **'beemStatic'** beemStatic network; extend generalized Lotka-Volterra model to cases of cross-sectional datasets to infer interaction among taxa based on expectation-maximization algorithm; see https://github.com/CSB5/BEEM-static for installing the R package; see Li et al. (2021) <doi:10.1371/journal.pcbi.1009343> for algorithm details

`COR_p_thres` default 0.01; the p value threshold for the correlation-based network.

`COR_p_adjust` default "fdr"; p value adjustment method, see method of p.adjust function for available options.

`COR_weight` default TRUE; whether use correlation coefficient as the weight of edges; FALSE represents weight = 1 for all edges.

`COR_cut` default 0.6; correlation coefficient threshold for the correlation network.

`COR_optimization` default FALSE; whether use random matrix theory (RMT) based method to determine the correlation coefficient; see https://doi.org/10.1186/1471-2105-13-113

`COR_optimization_low_high` default c(0.4, 0.8); the low and high value threshold used for the RMT optimization; only useful when COR_optimization = TRUE.

`SpiecEasi_method` default "mb"; either 'glasso' or 'mb';see spiec.easi function in package SpiecEasi and https://github.com/zdk123/SpiecEasi.

`FlashWeave_tempdir` default NULL; The temporary directory used to save the temporary files for running FlashWeave; If not assigned, use the system user temp.

`FlashWeave_meta_data` default FALSE; whether use env data for the optimization, If TRUE, the function automatically find the object$env_data in the object and generate a file for meta_data_path parameter of FlashWeave.

`FlashWeave_other_para` default "alpha=0.01,sensitive=true,heterogeneous=true"; the parameters used for FlashWeave; user can change the parameters or add more according to Flash-

Weave help document; An exception is meta_data_path parameter as it is generated based on the data inside the object, see FlashWeave_meta_data parameter for the description.

`beemStatic_t_strength` default 0.001; for network_method = "beemStatic"; the threshold used to limit the number of interactions (strength); same with the t.strength parameter in showInteraction function of beemStatic package.

`beemStatic_t_stab` default 0.8; for network_method = "beemStatic"; the threshold used to limit the number of interactions (stability); same with the t.stab parameter in showInteraction function of beemStatic package.

`add_taxa_name` default "Phylum"; one or more taxonomic rank name; used to add taxonomic rank name to network node properties.

`usename_rawtaxa_when_taxalevel_notOTU` default FALSE; whether replace the name of nodes using the taxonomic information.

`...` paremeters pass to spiec.easi function of SpiecEasi package when network_method = "SpiecEasi" or func.EM function of beemStatic package when network_method = "beemStatic".

Returns: res_network stored in object.

Examples:

```
\donttest{
# for correlation network
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1 <- trans_network$new(dataset = dataset, cal_cor = NA, filter_thres = 0.0001)
t1$cal_network(network_method = "SpiecEasi")
t1 <- trans_network$new(dataset = dataset, cal_cor = NA, filter_thres = 0.0001)
t1$cal_network(network_method = "FlashWeave")
}
```

Method `cal_module()`**:** Calculate network modules and add module names to the network node properties.

Usage:

```
trans_network$cal_module(
  method = "cluster_fast_greedy",
  module_name_prefix = "M"
)
```

Arguments:

`method` default "cluster_fast_greedy"; the method used to find the optimal community structure of a graph; the following are available functions (options) from igraph package: "cluster_fast_greedy", "cluster_optimal", "cluster_edge_betweenness", "cluster_infomap", "cluster_label_prop", "cluster_leading_eigen", "cluster_louvain", "cluster_spinglass", "cluster_walktrap". For the details of these functions, see the help document, such as help(cluster_fast_greedy); Note that the default "cluster_fast_greedy" method can only be used for undirected network. If the user selects network_method = "beemStatic" in cal_network function or provides other directed network, please use cluster_optimal or others for the modules identification.

`module_name_prefix` default "M"; the prefix of module names; module names are made of the module_name_prefix and numbers; numbers are assigned according to the sorting result of node numbers in modules with decreasing trend.

Returns: res_network with modules, stored in object.

Examples:

```
\donttest{
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1$cal_module(method = "cluster_fast_greedy")
}
```

Method `save_network()`**:** Save network as gexf style, which can be opened by Gephi (https://gephi.org/).

Usage:

```
trans_network$save_network(filepath = "network.gexf")
```

Arguments:

`filepath` default "network.gexf"; file path to save the network.

Returns: None.

Examples:

```
\dontrun{
t1$save_network(filepath = "network.gexf")
}
```

Method `cal_network_attr()`**:** Calculate network properties.

Usage:

```
trans_network$cal_network_attr()
```

Returns: res_network_attr stored in object.

Examples:

```
\donttest{
t1$cal_network_attr()
}
```

Method `cal_node_type()`**:** Calculate node properties. This function will be deprecated in the next release! Please use get_node_table function!

Usage:

```
trans_network$cal_node_type()
```

Returns: see the Return part in function get_node_table.

Method `get_node_table()`**:** Get the node property table. The properties may include the node names, modules allocation, degree, betweenness, abundance, taxonomy, within-module connectivity and among-module connectivity <doi:10.1016/j.geoderma.2022.115866>.

Authors: Chi Liu, Umer Zeeshan Ijaz

Usage:

```
trans_network$get_node_table(node_roles = TRUE)
```

Arguments:

node_roles default TRUE; whether calculate node roles, i.e. Module hubs, Network hubs, Connectors and Peripherals <doi:10.1016/j.geoderma.2022.115866>.

Returns: res_node_table in object; Abundance expressed as a percentage; z represents within-module connectivity; p represents among-module connectivity.

Examples:

```
\donttest{
t1$get_node_table(node_roles = TRUE)
}
```

Method `get_edge_table()`**:** Get the edge property table, including connected nodes, label and weight.

Usage:

```
trans_network$get_edge_table()
```

Returns: res_edge_table in object.

Examples:

```
\donttest{
t1$get_edge_table()
}
```

Method `get_adjacency_matrix()`**:** Get the adjacency matrix from the network graph.

Usage:

```
trans_network$get_adjacency_matrix(...)
```

Arguments:

`...` parameters passed to as_adjacency_matrix function of igraph package.

Returns: res_adjacency_matrix in object.

Examples:

```
\donttest{
t1$get_adjacency_matrix(attr = "weight")
}
```

Method `plot_network()`**:** Plot the network based on a series of methods from other packages, such as igraph, ggraph and networkD3. The networkD3 package provides dynamic network. It is especially useful for a glimpse of the whole network structure and finding the interested nodes and edges in a large network. In contrast, the igraph and ggraph methods are suitable for relatively small network.

Usage:

```
trans_network$plot_network(
  method = c("igraph", "ggraph", "networkD3")[1],
  node_label = "name",
  node_color = NULL,
  ggraph_layout = "fr",
  ggraph_node_size = 2,
  ggraph_text_color = NULL,
  ggraph_text_size = 3,
```

```
  networkD3_node_legend = TRUE,
  networkD3_zoom = TRUE,
  ...
)
```

Arguments:

`method` default "igraph"; The available options:

- **'igraph'** call plot.igraph function in igraph package for a static network; see plot.igraph for the parameters
- **'ggraph'** call ggraph function in ggraph package for a static network
- **'networkD3'** use forceNetwork function in networkD3 package for a dynamic network; see forceNetwork function for the parameters

`node_label` default "name"; node label shown in the plot for method = "ggraph" or method = "networkD3"; Please see the column names of object$res_node_table, which is the returned table of function object$get_node_table; User can select other column names in res_node_table.

`node_color` default NULL; node color assignment for method = "ggraph" or method = "networkD3"; Select a column name of object$res_node_table, such as "module".

`ggraph_layout` default "fr"; for method = "ggraph"; see layout parameter of create_layout function in ggraph package.

`ggraph_node_size` default 2; for method = "ggraph"; the node size.

`ggraph_text_color` default NULL; for method = "ggraph"; a column name of object$res_node_table; User can select other column names or change the content of object$res_node_table.

`ggraph_text_size` default 3; for method = "ggraph"; the node label text size.

`networkD3_node_legend` default TRUE; used for method = "networkD3"; logical value to enable node colour legends; Please see the legend parameter in networkD3::forceNetwork function.

`networkD3_zoom` default TRUE; used for method = "networkD3"; logical value to enable (TRUE) or disable (FALSE) zooming; Please see the zoom parameter in networkD3::forceNetwork function.

`...` parameters passed to plot.igraph function when method = "igraph" or forceNetwork function when method = "networkD3".

Returns: network plot.

Examples:

```
\donttest{
t1$plot_network(method = "igraph", layout = layout_with_kk)
t1$plot_network(method = "ggraph", node_color = "module")
t1$plot_network(method = "networkD3", node_color = "module")
}
```

Method `cal_eigen()`**:** Calculate eigengenes of modules, i.e. the first principal component based on PCA analysis, and the percentage of variance <doi:10.1186/1471-2105-13-113>.

Usage:

```
trans_network$cal_eigen()
```

Returns: res_eigen and res_eigen_expla in object.

Examples:

```
\donttest{
t1$cal_eigen()
}
```

Method `plot_taxa_roles()`**:** Plot the classification and importance of nodes, see object$res_node_table for the variable names used in the parameters.

Usage:

```
trans_network$plot_taxa_roles(
  use_type = c(1, 2)[1],
  roles_color_background = FALSE,
  roles_color_values = NULL,
  plot_module = FALSE,
  x_lim = c(0, 1),
  use_level = "Phylum",
  show_value = c("z", "p"),
  show_number = 1:10,
  plot_color = "Phylum",
  plot_shape = "taxa_roles",
  plot_size = "Abundance",
  color_values = RColorBrewer::brewer.pal(12, "Paired"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  ...
)
```

Arguments:

`use_type` default 1; 1 or 2; 1 represents taxa roles area plot; 2 represents the layered plot with taxa as x axis.

`roles_color_background` default FALSE; for use_type=1; TRUE: use background colors for each area; FALSE: use classic point colors.

`roles_color_values` default NULL; for use_type=1; color palette for background or points.

`plot_module` default FALSE; for use_type=1; whether plot the modules information.

`x_lim` default c(0, 1); for use_type=1; x axis range when roles_color_background = FALSE.

`use_level` default "Phylum"; for use_type=2; used taxonomic level in x axis.

`show_value` default c("z", "p"); for use_type=2; used variable in y axis.

`show_number` default 1:10; for use_type=2; showed number in x axis, sorting according to the nodes number.

`plot_color` default "Phylum"; for use_type=2; used variable for color.

`plot_shape` default "taxa_roles"; for use_type=2; used variable for shape.

`plot_size` default "Abundance"; for use_type=2; used for point size; a fixed number (e.g. 5) is also available.

`color_values` default RColorBrewer::brewer.pal(12, "Paired"); for use_type=2; color vector

`shape_values` default c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14); for use_type=2; shape vector, see ggplot2 tutorial for the shape meaning.

`...` paremeters pass to geom_point.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_taxa_roles(roles_color_background = FALSE)
}
```

Method `subset_network()`**:** Subset of the network.

Usage:

```
trans_network$subset_network(node = NULL, edge = NULL, rm_single = TRUE)
```

Arguments:

`node` default NULL; provide the node names that you want to use in the sub-network.

`edge` default NULL; provide the edge name needed; must be one of "+" or "-".

`rm_single` default TRUE; whether remove the nodes without any edge in the sub-network.

Returns: a new network

Examples:

```
\donttest{
t1$subset_network(node = t1$res_node_table %>% base::subset(module == "M1") %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1
}
```

Method `cal_powerlaw()`**:** Fit degrees to a power law distribution. First, perform a bootstrapping hypothesis test to determine whether degrees follow a power law distribution. If the distribution follows power law, then fit degrees to power law distribution and return the parameters.

Usage:

```
trans_network$cal_powerlaw(...)
```

Arguments:

`...` paremeters pass to fit_power_law function in igraph package.

Returns: res_powerlaw_p and res_powerlaw_fit; see bootstrap_p function in poweRlaw package for the bootstrapping p value details; see fit_power_law function in igraph package for the power law fit return details.

Examples:

```
\donttest{
t1$cal_powerlaw()
}
```

Method `trans_comm()`**:** Transform classifed features to community-like microtable object for further analysis, such as module-taxa table.

Usage:

```
trans_network$trans_comm(use_col = "module", abundance = TRUE)
```

Arguments:

`use_col` default "module"; which column to use as the 'community'; must be one of the name of res_node_table from function get_node_table.

`abundance` default TRUE; whether sum abundance of taxa. TRUE: sum the abundance for a taxon across all samples; FALSE: sum the frequency for a taxon across all samples.

Returns: a new microtable class.

Examples:

```
\donttest{
t2 <- t1$trans_comm(use_col = "module")
}
```

Method `print()`**:** Print the trans_network object.

Usage:

```
trans_network$print()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_network$new`
## ------------------------------------------------


data(dataset)
# for correlation network
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
# for other network
t1 <- trans_network$new(dataset = dataset, cal_cor = NA)


## ------------------------------------------------
## Method `trans_network$cal_network`
## ------------------------------------------------


# for correlation network
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1 <- trans_network$new(dataset = dataset, cal_cor = NA, filter_thres = 0.0001)
t1$cal_network(network_method = "SpiecEasi")
t1 <- trans_network$new(dataset = dataset, cal_cor = NA, filter_thres = 0.0001)
t1$cal_network(network_method = "FlashWeave")


## ------------------------------------------------
## Method `trans_network$cal_module`
## ------------------------------------------------
```

```
t1 <- trans_network$new(dataset = dataset, cal_cor = "base",
taxa_level = "OTU", filter_thres = 0.0001)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1$cal_module(method = "cluster_fast_greedy")


## ------------------------------------------------
## Method `trans_network$save_network`
## ------------------------------------------------

## Not run:
t1$save_network(filepath = "network.gexf")

## End(Not run)

## ------------------------------------------------
## Method `trans_network$cal_network_attr`
## ------------------------------------------------


t1$cal_network_attr()


## ------------------------------------------------
## Method `trans_network$get_node_table`
## ------------------------------------------------


t1$get_node_table(node_roles = TRUE)


## ------------------------------------------------
## Method `trans_network$get_edge_table`
## ------------------------------------------------


t1$get_edge_table()


## ------------------------------------------------
## Method `trans_network$get_adjacency_matrix`
## ------------------------------------------------


t1$get_adjacency_matrix(attr = "weight")


## ------------------------------------------------
## Method `trans_network$plot_network`
## ------------------------------------------------
```

```
t1$plot_network(method = "igraph", layout = layout_with_kk)
t1$plot_network(method = "ggraph", node_color = "module")
t1$plot_network(method = "networkD3", node_color = "module")


## ------------------------------------------------
## Method `trans_network$cal_eigen`
## ------------------------------------------------


t1$cal_eigen()


## ------------------------------------------------
## Method `trans_network$plot_taxa_roles`
## ------------------------------------------------


t1$plot_taxa_roles(roles_color_background = FALSE)


## ------------------------------------------------
## Method `trans_network$subset_network`
## ------------------------------------------------


t1$subset_network(node = t1$res_node_table %>% base::subset(module == "M1") %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1


## ------------------------------------------------
## Method `trans_network$cal_powerlaw`
## ------------------------------------------------


t1$cal_powerlaw()


## ------------------------------------------------
## Method `trans_network$trans_comm`
## ------------------------------------------------


t2 <- t1$trans_comm(use_col = "module")
```

`trans_nullmodel` *Create trans_nullmodel object for phylogeny- and taxonomy-based null model analysis.*

Description

This class is a wrapper for a series of null model related approaches, including the mantel correlogram analysis of phylogenetic signal, beta nearest taxon index (betaNTI), beta net relatedness index (betaNRI), NTI, NRI and RCbray calculations; See Stegen et al. (2013) <10.1038/ismej.2013.93> and Liu et al. (2017) <doi:10.1038/s41598-017-17736-w> for the algorithms and applications.

Methods

Public methods:

- `trans_nullmodel$new()`
- `trans_nullmodel$cal_mantel_corr()`
- `trans_nullmodel$plot_mantel_corr()`
- `trans_nullmodel$cal_betampd()`
- `trans_nullmodel$cal_betamntd()`
- `trans_nullmodel$cal_ses_betampd()`
- `trans_nullmodel$cal_ses_betamntd()`
- `trans_nullmodel$cal_rcbray()`
- `trans_nullmodel$cal_process()`
- `trans_nullmodel$cal_NRI()`
- `trans_nullmodel$cal_NTI()`
- `trans_nullmodel$cal_Cscore()`
- `trans_nullmodel$cal_tNST()`
- `trans_nullmodel$cal_tNST_test()`
- `trans_nullmodel$clone()`

Method `new()`:

Usage:

```
trans_nullmodel$new(
  dataset = NULL,
  filter_thres = 0,
  taxa_number = NULL,
  group = NULL,
  select_group = NULL,
  env_cols = NULL,
  add_data = NULL,
  complete_na = FALSE
)
```

Arguments:

`dataset` the object of `microtable` Class.

`filter_thres` default 0; the relative abundance threshold.

`taxa_number` default NULL; how many taxa the user want to keep, if provided, filter_thres parameter will be forcible invalid.

`group` default NULL; which group column name in sample_table is selected.

`select_group` default NULL; the group name, used following the group to filter samples.

env_cols default NULL; number or name vector to select the environmental data in dataset$sample_table.

add_data default NULL; provide environmental data table additionally.

complete_na default FALSE; whether fill the NA in environmental data based on the method in mice package.

Returns: data_comm and data_tree in object.

Examples:

```
data(dataset)
data(env_data_16S)
t1 <- trans_nullmodel$new(dataset, filter_thres = 0.0005, add_data = env_data_16S)
```

Method `cal_mantel_corr()`**:** Calculate mantel correlogram.

Usage:

```
trans_nullmodel$cal_mantel_corr(
  use_env = NULL,
  break.pts = seq(0, 1, 0.02),
  cutoff = FALSE,
  ...
)
```

Arguments:

use_env default NULL; numeric or character vector to select env_data; if provide multiple variables or NULL, use PCA (principal component analysis) to reduce dimensionality.

break.pts default seq(0, 1, 0.02); see break.pts parameter in `mantel.correlog` of vegan package.

cutoff default FALSE; see cutoff parameter in `mantel.correlog`.

... parameters pass to `mantel.correlog`

Returns: res_mantel_corr in object.

Examples:

```
\donttest{
t1$cal_mantel_corr(use_env = "pH")
}
```

Method `plot_mantel_corr()`**:** Plot mantel correlogram.

Usage:

```
trans_nullmodel$plot_mantel_corr(point_shape = 22, point_size = 3)
```

Arguments:

point_shape default 22; the number for selecting point shape type; see ggplot2 manual for the number meaning.

point_size default 3; the point size.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_mantel_corr()
}
```

Method `cal_betampd()`**:** Calculate betaMPD (mean pairwise distance). Same with comdist in picante package, but faster.

Usage:

```
trans_nullmodel$cal_betampd(abundance.weighted = TRUE)
```

Arguments:

`abundance.weighted` default TRUE; whether use abundance-weighted method.

Returns: res_betampd in object.

Examples:

```
\donttest{
t1$cal_betampd(abundance.weighted = TRUE)
}
```

Method `cal_betamntd()`**:** Calculate betaMNTD (mean nearest taxon distance). Same with comdistnt in picante package, but faster.

Usage:

```
trans_nullmodel$cal_betamntd(
  abundance.weighted = TRUE,
  exclude.conspecifics = FALSE,
  use_iCAMP = FALSE,
  use_iCAMP_force = TRUE,
  iCAMP_tempdir = NULL,
  ...
)
```

Arguments:

`abundance.weighted` default TRUE; whether use abundance-weighted method.

`exclude.conspecifics` default FALSE; see exclude.conspecifics parameter in comdistnt function of picante package.

`use_iCAMP` default FALSE; whether use bmntd.big function of iCAMP package to calculate betaMNTD. This method can store the phylogenetic distance matrix on the disk to lower the memory spending and perform the calculation parallelly.

`use_iCAMP_force` default FALSE; whether use bmntd.big function of iCAMP package automatically when the feature number is large.

`iCAMP_tempdir` default NULL; the temporary directory used to place the large tree file; If NULL; use the system user tempdir.

`...` paremeters pass to iCAMP::pdist.big function.

Returns: res_betamntd in object.

Examples:

```
\donttest{
t1$cal_betamntd(abundance.weighted = TRUE)
}
```

Method `cal_ses_betampd()`**:** Calculate standardized effect size of betaMPD, i.e. beta net relatedness index (betaNRI).

Usage:

```
trans_nullmodel$cal_ses_betampd(
  runs = 1000,
  null.model = c("taxa.labels", "richness", "frequency", "sample.pool",
    "phylogeny.pool", "independentswap", "trialswap")[1],
  abundance.weighted = TRUE,
  iterations = 1000
)
```

Arguments:

`runs` default 1000; simulation runs.

`null.model` default "taxa.labels"; The available options include "taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool", "independentswap"and "trialswap"; see null.model parameter of ses.mntd function in picante package for the algorithm details.

`abundance.weighted` default TRUE; whether use weighted abundance.

`iterations` default 1000; iteration number for part null models to perform; see iterations parameter of picante::randomizeMatrix function.

Returns: res_ses_betampd in object.

Examples:

```
\donttest{
# run 50 times for the example; default 1000
t1$cal_ses_betampd(runs = 50, abundance.weighted = TRUE)
}
```

Method `cal_ses_betamntd()`**:** Calculate standardized effect size of betaMNTD, i.e. beta nearest taxon index (betaNTI).

Usage:

```
trans_nullmodel$cal_ses_betamntd(
  runs = 1000,
  null.model = c("taxa.labels", "richness", "frequency", "sample.pool",
    "phylogeny.pool", "independentswap", "trialswap")[1],
  abundance.weighted = TRUE,
  exclude.conspecifics = FALSE,
  use_iCAMP = FALSE,
  use_iCAMP_force = TRUE,
  iCAMP_tempdir = NULL,
  nworker = 2,
  iterations = 1000
)
```

Arguments:

`runs` default 1000; simulation number of null model.

`null.model` default "taxa.labels"; The available options include "taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool", "independentswap"and "trialswap"; see null.model parameter of ses.mntd function in picante package for the algorithm details.

`abundance.weighted` default TRUE; whether use abundance-weighted method.

`exclude.conspecifics` default FALSE; see comdistnt in picante package.

use_iCAMP default FALSE; whether use bmntd.big function of iCAMP package to calculate betaMNTD. This method can store the phylogenetic distance matrix on the disk to lower the memory spending and perform the calculation parallelly.

use_iCAMP_force default FALSE; whether to make use_iCAMP to be TRUE when the feature number is large.

iCAMP_tempdir default NULL; the temporary directory used to place the large tree file; If NULL; use the system user tempdir.

nworker default 2; the CPU thread number.

iterations default 1000; iteration number for part null models to perform; see iterations parameter of picante::randomizeMatrix function.

Returns: res_ses_betamntd in object.

Examples:

```
\donttest{
# run 50 times for the example; default 1000
t1$cal_ses_betamntd(runs = 50, abundance.weighted = TRUE, exclude.conspecifics = FALSE)
}
```

Method `cal_rcbray()`**:** Calculate Bray–Curtis-based Raup–Crick (RCbray).

Usage:

```
trans_nullmodel$cal_rcbray(
  runs = 1000,
  verbose = TRUE,
  null.model = "independentswap"
)
```

Arguments:

runs default 1000; simulation runs.

verbose default TRUE; whether show the calculation process message.

null.model default "independentswap"; see more available options in randomizeMatrix function of picante package.

Returns: res_rcbray in object.

Examples:

```
\donttest{
# run 50 times for the example; default 1000
t1$cal_rcbray(runs = 50)
}
```

Method `cal_process()`**:** Infer the ecological processes according to ses.betaMNTD ses.betaMPD and rcbray.

Usage:

```
trans_nullmodel$cal_process(use_betamntd = TRUE)
```

Arguments:

use_betamntd default TRUE; whether use ses.betaMNTD; if false, use ses.betaMPD.

Returns: res_rcbray in object.

Examples:

```
\donttest{
t1$cal_process(use_betamntd = TRUE)
}
```

Method `cal_NRI()`**:** Calculates Nearest Relative Index (NRI), equivalent to -1 times the standardized effect size of MPD.

Usage:

```
trans_nullmodel$cal_NRI(
  null.model = "taxa.labels",
  abundance.weighted = FALSE,
  runs = 999,
  ...
)
```

Arguments:

`null.model` default "taxa.labels"; Null model to use; see null.model parameter in ses.mpd function of picante package for available options.

`abundance.weighted` default FALSE; Should mean nearest relative distances for each species be weighted by species abundance?

`runs` default 999; Number of randomizations.

`...` paremeters pass to ses.mpd function in picante package.

Returns: res_NRI in object, equivalent to -1 times ses.mpd.

Examples:

```
\donttest{
t1$cal_NRI()
}
```

Method `cal_NTI()`**:** Calculates Nearest Taxon Index (NTI), equivalent to -1 times the standardized effect size of MNTD.

Usage:

```
trans_nullmodel$cal_NTI(
  null.model = "taxa.labels",
  abundance.weighted = FALSE,
  runs = 999,
  ...
)
```

Arguments:

`null.model` default "taxa.labels"; Null model to use; see null.model parameter in ses.mntd function of picante package for available options.

`abundance.weighted` default FALSE; Should mean nearest taxon distances for each species be weighted by species abundance?

`runs` default 999; Number of randomizations.

`...` paremeters pass to ses.mntd function in picante package.

Returns: res_NTI in object, equivalent to -1 times ses.mntd.

Examples:

```
\donttest{
t1$cal_NTI(null.model = "taxa.labels", abundance.weighted = TRUE)
}
```

Method `cal_Cscore()`**:** Calculates the (normalised) mean number of checkerboard combinations (C-score) using C.score function in bipartite package.

Usage:

```
trans_nullmodel$cal_Cscore(by_group = NULL, ...)
```

Arguments:

`by_group` default NULL; one column name or number in sample_table; calculate C-score for different groups separately.

`...` paremeters pass to C.score function in bipartite package.

Returns: results directly.

Examples:

```
\donttest{
t1$cal_Cscore()
}
```

Method `cal_tNST()`**:** Calculate normalized stochasticity ratio (NST) based on the tNST function of NST package.

Usage:

```
trans_nullmodel$cal_tNST(group, ...)
```

Arguments:

`group` a colname of sample_table; the function can select the data from sample_table to generate a one-column (n x 1) matrix and provide it to the group parameter of tNST function.

`...` paremeters pass to tNST function of NST package; see the documents of tNST function for more details.

Returns: .

Examples:

```
\donttest{
t1$cal_tNST(group = "Group", dist.method = "bray", output.rand = TRUE, SES = TRUE)
}
```

Method `cal_tNST_test()`**:** Test the significance of NST difference between each pair of groups.

Usage:

```
trans_nullmodel$cal_tNST_test(method = "nst.boot", ...)
```

Arguments:

`method` default "nst.boot"; "nst.boot" or "nst.panova"; see NST::nst.boot function or NST::nst.panova function for the details.

`...` paremeters pass to NST::nst.boot when method = "nst.boot" or NST::nst.panova when method = "nst.panova"

Returns: .

Examples:

```
\donttest{
t1$cal_tNST_test()
}
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_nullmodel$new`
## ------------------------------------------------

data(dataset)
data(env_data_16S)
t1 <- trans_nullmodel$new(dataset, filter_thres = 0.0005, add_data = env_data_16S)

## ------------------------------------------------
## Method `trans_nullmodel$cal_mantel_corr`
## ------------------------------------------------


t1$cal_mantel_corr(use_env = "pH")


## ------------------------------------------------
## Method `trans_nullmodel$plot_mantel_corr`
## ------------------------------------------------


t1$plot_mantel_corr()


## ------------------------------------------------
## Method `trans_nullmodel$cal_betampd`
## ------------------------------------------------


t1$cal_betampd(abundance.weighted = TRUE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_betamntd`
## ------------------------------------------------
```

```
t1$cal_betamntd(abundance.weighted = TRUE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_ses_betampd`
## ------------------------------------------------


# run 50 times for the example; default 1000
t1$cal_ses_betampd(runs = 50, abundance.weighted = TRUE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_ses_betamntd`
## ------------------------------------------------


# run 50 times for the example; default 1000
t1$cal_ses_betamntd(runs = 50, abundance.weighted = TRUE, exclude.conspecifics = FALSE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_rcbray`
## ------------------------------------------------


# run 50 times for the example; default 1000
t1$cal_rcbray(runs = 50)


## ------------------------------------------------
## Method `trans_nullmodel$cal_process`
## ------------------------------------------------


t1$cal_process(use_betamntd = TRUE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_NRI`
## ------------------------------------------------


t1$cal_NRI()


## ------------------------------------------------
## Method `trans_nullmodel$cal_NTI`
## ------------------------------------------------


t1$cal_NTI(null.model = "taxa.labels", abundance.weighted = TRUE)
```

```
## ------------------------------------------------
## Method `trans_nullmodel$cal_Cscore`
## ------------------------------------------------


t1$cal_Cscore()


## ------------------------------------------------
## Method `trans_nullmodel$cal_tNST`
## ------------------------------------------------


t1$cal_tNST(group = "Group", dist.method = "bray", output.rand = TRUE, SES = TRUE)


## ------------------------------------------------
## Method `trans_nullmodel$cal_tNST_test`
## ------------------------------------------------


t1$cal_tNST_test()
```

`trans_venn` *Create trans_venn object.*

Description

This class is a wrapper for a series of venn analysis related methods, including venn result, 2- to 5-way venn diagram, more than 5-way petal plot and venn result transformations based on David et al. (2012) <doi:10.1128/AEM.01459-12>.

Methods

Public methods:

- `trans_venn$new()`
- `trans_venn$plot_venn()`
- `trans_venn$trans_comm()`
- `trans_venn$print()`
- `trans_venn$clone()`

Method `new()`:

Usage:

```
trans_venn$new(
  dataset = NULL,
  sample_names = NULL,
  ratio = NULL,
  add_abund_table = NULL
)
```

Arguments:

`dataset` the object of `microtable` Class.

`sample_names` default NULL; if provided, filter the samples.

`ratio` default NULL; NULL, "numratio" or "seqratio"; numratio: calculate number percentage; seqratio: calculate sequence percentage; NULL: no additional percentage.

`add_abund_table` default NULL; data.frame or matrix format; additional data provided instead of dataset$otu_table; Features must be rows.

Returns: data_details and data_summary stored in trans_venn object.

Examples:

```
\donttest{
data(dataset)
t1 <- dataset$merge_samples(use_group = "Group")
t1 <- trans_venn$new(dataset = t1, ratio = "numratio")
}
```

Method `plot_venn()`**:** Plot venn diagram.

Usage:

```
trans_venn$plot_venn(
  color_circle = RColorBrewer::brewer.pal(8, "Dark2"),
  fill_color = TRUE,
  text_size = 4.5,
  text_name_size = 6,
  text_name_position = NULL,
  alpha = 0.3,
  linesize = 1.1,
  petal_plot = FALSE,
  petal_color = "#BEAED4",
  petal_color_center = "#BEBADA",
  petal_a = 4,
  petal_r = 1,
  petal_use_lim = c(-12, 12),
  petal_center_size = 40,
  petal_move_xy = 4,
  petal_move_k = 2.3,
  petal_move_k_count = 1.3,
  petal_text_move = 40,
  other_text_show = NULL,
  other_text_position = c(2, 2),
  other_text_size = 5
)
```

Arguments:

`color_circle` default RColorBrewer::brewer.pal(8, "Dark2"); color pallete

`fill_color` default TRUE; whether fill the area color

`text_size` default 4.5; text size in plot

`text_name_size` default 6; name size in plot

`text_name_position` default NULL; name position in plot

`alpha` default .3; alpha for transparency

`linesize` default 1.1; cycle line size

`petal_plot` default FALSE; whether use petal plot.

`petal_color` default "#BEAED4"; color of the petals.

`petal_color_center` default "#BEBADA"; color of the center in the petal plot.

`petal_a` default 4; the length of the ellipse

`petal_r` default 1; scaling up the size of the ellipse

`petal_use_lim` default c(-12, 12); the width of the plot

`petal_center_size` default 40; petal center circle size

`petal_move_xy` default 4; the distance of text to circle

`petal_move_k` default 2.3; the distance of title to circle

`petal_move_k_count` default 1.3; the distance of data text to circle

`petal_text_move` default 40; the distance between two data text

`other_text_show` default NULL; other characters used to show in the plot

`other_text_position` default c(1, 1); the text position for text in other_text_show

`other_text_size` default 5; the text size for text in other_text_show

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_venn()
}
```

Method `trans_comm()`**:** Transform venn result to community-like microtable object for further composition analysis.

Usage:

```
trans_venn$trans_comm(use_frequency = TRUE)
```

Arguments:

`use_frequency` default TRUE; whether only use OTUs occurrence frequency, i.e. presence/absence data; if FALSE, use abundance data.

Returns: a new `microtable` class.

Examples:

```
\donttest{
t2 <- t1$trans_comm(use_frequency = TRUE)
}
```

Method `print()`**:** Print the trans_venn object.

Usage:

```
trans_venn$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_venn$new`
## ------------------------------------------------


data(dataset)
t1 <- dataset$merge_samples(use_group = "Group")
t1 <- trans_venn$new(dataset = t1, ratio = "numratio")


## ------------------------------------------------
## Method `trans_venn$plot_venn`
## ------------------------------------------------


t1$plot_venn()


## ------------------------------------------------
## Method `trans_venn$trans_comm`
## ------------------------------------------------


t2 <- t1$trans_comm(use_frequency = TRUE)
```

Index