Package ‘microeco’

November 16, 2021

Type Package

Title Microbial Community Ecology Data Analysis

Version 0.6.0

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 statistics, beta diversity statistics, differential abundance test and indicator taxon analysis, environmental data analysis, null model analysis, network analysis and functional analysis.

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,
tidytree, mice, ggtree

License GPL-3

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2021-11-16 09:10:02 UTC

RoxygenNote 7.1.1

R topics documented:

clone 2
dataset 3
dropallfactors 4
env_data_16S 4
fungi_func_FungalTraits 5
fungi_func_FUNGuild 5
ko_map 5
microtable 6
otu_table_16S 15
otu_table_ITS 15
phylo_tree_16S 16
prok_func_FAPROTAX 16
prok_func_NJC19_list 16
rep_fasta_16S 16
sample_info_16S 17
sample_info_ITS 17
Tax4Fun2_KEGG 17
taxonomy_table_16S 17
taxonomy_table_ITS 18
tidy_taxonomy 18
trans_abund 19
trans_alpha 25
trans_beta 28
trans_diff 34
trans_env 39
trans_func 48
trans_network 54
trans_nullmodel 62
trans_venn 68

Index **71**

`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)
```

`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)
```

`ko_map` *The KEGG pathway annotation database in the microeco package*

Description

The KEGG pathway annotation database in the microeco package

Usage

```
data(ko_map)
```

`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.

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
)
```

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; default NULL; The representative sequences; use read.fasta function in seqinr package for input.

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 = TRUE)
```

Arguments:

`main_data` TRUE or 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` *Clear up the taxonomic table to make taxonomic assignments consistent.*

Description

Clear up the taxonomic table to make taxonomic assignments consistent.

Usage

```
tidy_taxonomy(taxonomy_table)
```

Arguments

`taxonomy_table`	a data.frame with taxonomic information.

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 approaches include the bar plot, boxplot, heatmap and pie chart based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>.

Methods

Public methods:

- `trans_abund$new()`
- `trans_abund$plot_bar()`
- `trans_abund$plot_heatmap()`
- `trans_abund$plot_box()`
- `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; microtable object.

`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 and the character in front of them.

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

`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.

`use_percentage` default TRUE; show the abundance percentage.

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

Returns: abund_data for plotting.

Examples:

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

Method `plot_bar()`**:** Bar plot in trans_abund object.

Usage:

```
trans_abund$plot_bar(
  use_colors = RColorBrewer::brewer.pal(12, "Paired"),
  bar_type = "full",
  others_color = "grey90",
  facet = 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,
  base_font = NULL,
  ylab_title = NULL
)
```

Arguments:

`use_colors` default RColorBrewer::brewer.pal(12, "Paired"); providing the plotting colors.

`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; a character string; if using facet, providing a group column name of 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").

`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.

`base_font` default NULL; ggplot font family in the plot.

`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 in trans_abund object.

Usage:

```
trans_abund$plot_heatmap(
 use_colors = c("#00008B", "#102D9B", "#215AAC", "#3288BD", "#66C2A5", "#E6F598",
   "#FFFFBF", "#FED690", "#FDAE61", "#F46D43", "#D53E4F", "#9E0142"),
  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,
  base_font = NULL
)
```

Arguments:

`use_colors` default RColorBrewer::brewer.pal(12, "Paired"); providing the plotting colors.

`facet` default NULL; a character string; if using facet, providing a group column name of 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.

`base_font` default NULL; font in the plot.

Returns: ggplot2 plot.

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 in trans_abund object.

Usage:

```
trans_abund$plot_box(
  use_colors = 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,
  base_font = NULL,
  ...
)
```

Arguments:

`use_colors` default RColorBrewer::brewer.pal(12, "Paired"); providing the plotting colors.

`group` default NULL; 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.

`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.

`base_font` default NULL; font in the plot.

`...` parameters pass to `geom_boxplot`.

Returns: ggplot2 plot.

Examples:

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

Method `plot_pie()`: Plot pie chart in trans_abund class.

Usage:

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

Arguments:

`use_colors` default RColorBrewer::brewer.pal(8, "Dark2"); providing the plotting colors.

`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_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 alpha_stat.

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

Returns: alpha_data and alpha_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", "anova")[1],
  measures = NULL,
  anova_set = NULL
)
```

Arguments:

`method` default "KW"; "KW" or "anova"; KW rank sum test or anova for the testing.

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

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

Returns: res_alpha_diff in object. A data.frame for method = 'KW' or 'anova'. A list for method = 'anova' and anova_set is assigned.

Examples:

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

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

Usage:

```
trans_alpha$plot_alpha(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  measure = "Shannon",
  group = NULL,
  add_letter = FALSE,
  use_boxplot = TRUE,
  boxplot_color = TRUE,
  boxplot_add = "jitter",
  order_x_mean = TRUE,
  pair_compare = FALSE,
  pair_compare_filter = "",
  pair_compare_method = "wilcox.test",
  xtext_type = NULL,
  xtext_size = 10,
  ytitle_size = 17,
  base_font = "sans",
  ...
)
```

Arguments:

`color_values` colors used for presentation.

`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_letter` default FALSE; If TRUE, the letters of duncan test will be added in the plot.

`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.

`pair_compare` default FALSE; whether perform paired comparisons.

`pair_compare_filter` default ""; groups that need to be removed in the comparisons.

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

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

`xtext_size` default 10, x axis text size.

`ytitle_size` default 17, y axis title size.

`base_font` default "sans", font in the plot.

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

Returns: ggplot.

Examples:

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

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 = "anova")


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


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

`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.

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$print()
- trans_beta$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".

`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(
  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,
  plot_point_size = 3,
  plot_point_alpha = 0.9,
  plot_sample_label = NULL,
  plot_group_centroid = FALSE,
  plot_group = NULL,
  segment_alpha = 0.6,
  centroid_linetype = 3,
  plot_group_ellipse = FALSE,
  ellipse_level = 0.9,
  ellipse_alpha = 0.1,
  ellipse_type = "t"
)
```

Arguments:

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

`shape_values` default c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14); a vector used in the shape type, see ggplot2 tutorial.

`plot_color` default NULL; the sample group name used for color in plot.

`plot_shape` default NULL; the sample group name used for shape in plot.

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

`plot_point_size` default 3; point size in plot.

`plot_point_alpha` default .9; point transparency in plot.

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

`plot_group_centroid` default FALSE; whether show the centroid in each group of plot.

`plot_group` default NULL; the column name in sample table, generally used with plot_group_centroid and plot_group_ellipse.

`segment_alpha` default .6; segment transparency in plot.

`centroid_linetype` default 3; the line type related with centroid in plot.

`plot_group_ellipse` default FALSE; whether show the confidence ellipse in each group of plot.

`ellipse_level` default .9; confidence level of ellipse.

`ellipse_alpha` default .1; color transparency in the ellipse.

`ellipse_type` default t; see type in `stat_ellipse`.

Returns: ggplot.

Examples:

```
t1$plot_ordination(plot_color = "Group", plot_shape = "Group", plot_group_ellipse = TRUE)
```

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

Usage:

```
trans_beta$cal_manova(
  cal_manova_all = FALSE,
  cal_manova_paired = FALSE,
  cal_manova_set = NULL,
  permutations = 999
)
```

Arguments:

`cal_manova_all` default FALSE; whether manova is used for all data.

`cal_manova_paired` default FALSE; whether manova is used for all the paired groups.

`cal_manova_set` default NULL; specified group set for manova, see `adonis`.

`permutations` default 999; see permutations in `adonis`.

Returns: res_manova stored in object.

Examples:

```
t1$cal_manova(cal_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,
  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.

`pair_compare_filter_match` default NULL; if provided, remove the matched groups; use the regular express to match the paired groups.

`pair_compare_filter_select` default NULL; numeric vector; 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)
}
```

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 `print()`**:** Print the trans_beta object.

Usage:

```
trans_beta$print()
```

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_color = "Group", plot_shape = "Group", plot_group_ellipse = TRUE)

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

t1$cal_manova(cal_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)


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

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

`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 non-parametric test, LEfSe based on the Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>, random forest, metastat based on White et al. (2009) <doi:10.1371/journal.pcbi.1000352> and the method in R package metagenomeSeq Paulson et al. (2013) <doi:10.1038/nmeth.2658>.

Methods

Public methods:

- `trans_diff$new()`
- `trans_diff$plot_diff_abund()`
- `trans_diff$plot_lefse_bar()`
- `trans_diff$plot_lefse_cladogram()`
- `trans_diff$plot_metastat()`
- `trans_diff$print()`
- `trans_diff$clone()`

Method `new()`:

Usage:

```
trans_diff$new(
  dataset = NULL,
  method = c("lefse", "rf", "metastat", "mseq")[1],
  group = NULL,
  lefse_subgroup = NULL,
  alpha = 0.05,
  lefse_min_subsam = 10,
  lefse_norm = 1e+06,
  nresam = 0.6667,
  boots = 30,
  rf_taxa_level = "all",
  rf_ntree = 1000,
  metastat_taxa_level = "Genus",
  group_choose_paired = NULL,
  mseq_adjustMethod = "fdr",
  mseq_count = 1
)
```

Arguments:

`dataset` the object of `microtable` Class.

`method` default "lefse"; "lefse", "rf", "metastat" or "mseq". "lefse": Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>; "rf" represents random forest; metastat: White et al. (2009) <doi:10.1371/journal.pcbi.1000352>; "mseq" represents the method in metagenomeSeq package.

`group` default NULL; sample group used for main comparision.

`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>.

`alpha` default .05; significance threshold.

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

`lefse_norm` default 1000000; scale value in lefse.

`nresam` default .6667; sample number ratio used in each bootstrap or LEfSe or random forest.

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

`rf_taxa_level` default "all"; use all taxonomic rank data, if want to test a specific rank, provide taxonomic rank name, such as "Genus".

`rf_ntree` default 1000; see ntree in randomForest function of randomForest package.

`metastat_taxa_level` default "Genus"; taxonomic rank level used in metastat test; White et al. (2009) <doi:10.1371/journal.pcbi.1000352>.

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

`mseq_adjustMethod` default "fdr"; Method to adjust p-values by. Default is "fdr". Options include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none".

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

Returns: res_rf, res_lefse, res_abund, res_metastat, or res_mseq in trans_diff object, depending on the method.

Examples:

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

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

Usage:

```
trans_diff$plot_diff_abund(
  method = NULL,
  only_abund_plot = TRUE,
  use_number = 1:10,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  plot1_bar_color = "grey50",
  plot2_sig_color = "red",
  plot2_sig_size = 1.2,
  axis_text_y = 10,
  simplify_names = TRUE,
  keep_prefix = TRUE,
  group_order = NULL,
  plot2_barwidth = 0.9,
  add_significance = TRUE,
  use_se = TRUE
)
```

Arguments:

`method` default NULL; "rf" or "lefse"; automatically check the method in the result.

`only_abund_plot` default TRUE; if true, return only abundance plot; if false, return both indicator plot and abundance plot

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

`color_values` colors for presentation.

`plot1_bar_color` default "grey30"; the color for the plot 1.

`plot2_sig_color` default "red"; the color for the significance in plot 2.

`plot2_sig_size` default 1.5; the size for the significance in plot 2.

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

`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 in plot.

`plot2_barwidth` default .9; the bar width in plot 2.

`add_significance` default TRUE; whether add the significance asterisk; only available when only_abund_plot FALSE.

`use_se` default TRUE; whether use SE in plot 2, if FALSE, use SD.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_diff_abund(use_number = 1:10)
}
```

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

Usage:

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

Arguments:

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

`color_values` colors for presentation.

`LDA_score` default NULL; numeric value as the threshold, such as 2, limited with use_number.

`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 in plot.

`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_lefse_bar(LDA_score = 4)
}
```

Method `plot_lefse_cladogram()`**:** Plot the cladogram for LEfSe result similar with the python version. Codes are modified from microbiomeMarker

Usage:

```
trans_diff$plot_lefse_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 = 0.7,
  node_size_scale = 1,
  node_size_offset = 1,
  annotation_shape = 22,
  annotation_shape_size = 5
)
```

Arguments:

`color` default RColorBrewer::brewer.pal(8, "Dark2"); color 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

`group_order` default NULL; a vector to order the legend in plot.

`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 0.7; size for the clade label

`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_lefse_cladogram(use_taxa_num = 100, use_feature_num = 30, select_show_labels = NULL)
}
```

Method `plot_metastat()`**:** Bar plot for metastat.

Usage:

```
trans_diff$plot_metastat(
  use_number = 1:10,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  qvalue = 0.05,
  choose_group = 1
)
```

Arguments:

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

`color_values` colors for presentation.

`qvalue` default .05; numeric value as the threshold of q value.

`choose_group` default 1; which column in res_metastat_group_matrix will be used.

Returns: ggplot.

Examples:

```
\donttest{
t1 <- trans_diff$new(dataset = dataset, method = "metastat", group = "Group")
t1$plot_metastat(use_number = 1:10, qvalue = 0.05, choose_group = 1)
}
```

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

Usage:

```
trans_diff$print()
```

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")


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


t1$plot_diff_abund(use_number = 1:10)


## ------------------------------------------------
## Method `trans_diff$plot_lefse_bar`
## ------------------------------------------------


t1$plot_lefse_bar(LDA_score = 4)


## ------------------------------------------------
## Method `trans_diff$plot_lefse_cladogram`
## ------------------------------------------------


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


## ------------------------------------------------
## Method `trans_diff$plot_metastat`
## ------------------------------------------------


t1 <- trans_diff$new(dataset = dataset, method = "metastat", group = "Group")
t1$plot_metastat(use_number = 1:10, qvalue = 0.05, choose_group = 1)
```

`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 and correlation analysis based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>.

Methods

Public methods:

- `trans_env$new()`
- `trans_env$cal_rda()`
- `trans_env$cal_rda_envsquare()`
- `trans_env$trans_rda()`
- `trans_env$plot_rda()`
- `trans_env$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; a vector to select columns in sample_table, when the environmental data is in sample_table. Either numeric vector or character vector of colnames.

`add_data` default NULL; data.frame format; provide the environmental data frame individually.

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

`complete_na` default FALSE; Whether fill the NA in the environmental data.

Returns: env_data in trans_env object.

Examples:

```
data(dataset)
data(env_data_16S)
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S)
```

Method `cal_rda()`**:** Redundancy analysis (RDA) based on the rda function in vegan package.

Usage:

```
trans_env$cal_rda(
  use_dbrda = TRUE,
  add_matrix = NULL,
  use_measure = NULL,
  feature_sel = FALSE,
  taxa_level = NULL,
  taxa_filter_thres = NULL
)
```

Arguments:

`use_dbrda` default TRUE; whether use db-RDA, if FALSE, use RDA.

`add_matrix` default NULL; additional distance matrix provided, if you do not want to use the beta diversity matrix within the dataset.

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

`feature_sel` default FALSE; whether perform the feature selection.

`taxa_level` default NULL; If use RDA, provide the taxonomic rank.

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

Returns: res_rda, res_rda_R2, res_rda_terms and res_rda_axis in object.

Examples:

```
\donttest{
t1$cal_rda(use_dbrda = TRUE, use_measure = "bray")
}
```

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

Usage:

```
trans_env$cal_rda_envsquare(...)
```

Arguments:

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

Returns: res_rda_envsquare in object.

Examples:

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

Method `trans_rda()`**:** transform RDA result for the following plotting.

Usage:

```
trans_env$trans_rda(
  show_taxa = 10,
  adjust_arrow_length = FALSE,
  min_perc_env = 1,
  max_perc_env = 100,
  min_perc_tax = 1,
  max_perc_tax = 100
)
```

Arguments:

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

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

`min_perc_env` default 1; minimum scale value for env arrow, relatively.

`max_perc_env` default 100; maximum scale value for env arrow, relatively.

`min_perc_tax` default 1; minimum scale value for tax arrow, relatively.

`max_perc_tax` default 100; maximum scale value for tax arrow, relatively.

Returns: res_rda_trans in object.

Examples:

```
\donttest{
t1$trans_rda(adjust_arrow_length = TRUE, max_perc_env = 10)
}
```

Method `plot_rda()`**:** plot RDA result.

Usage:

```
trans_env$plot_rda(
  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),
  taxa_text_color = "firebrick1",
  taxa_text_type = "italic"
)
```

Arguments:

`plot_color` default NULL; group used for color.

`plot_shape` default NULL; group used for shape.

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

`shape_values` default see the function; vector used in the shape, see ggplot2 tutorial.

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

`taxa_text_type` default "italic"; taxa text style; better to use "italic" for Genus, use "normal" for others.

Returns: ggplot object.

Examples:

```
\donttest{
t1$plot_rda(plot_color = "Group")
}
```

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",
  ...
)
```

Arguments:

`select_env_data` default NULL; numeric or character vector to select columns in env_data; 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.

`...` paremeters pass to `mantel`.

Returns: res_mantel in object.

Examples:

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

Method `cal_cor()`**:** Calculating the correlations between taxa abundance and environmental variables. Indeed, 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 env_data; 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.

`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; provide additional taxa, see use_data parameter.

`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; will retain samples within the group.

`taxa_name_full` default TRUE; Whether retain 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_rf$Taxa[1:40])
}
```

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

Usage:

```
trans_env$plot_cor(
  color_vector = c("#00008B", "#102D9B", "#215AAC", "#3288BD", "#66C2A5", "#E6F598",
    "#FFFFBF", "#FED690", "#FDAE61", "#F46D43", "#D53E4F"),
  pheatmap = FALSE,
  filter_feature = NULL,
  ylab_type_italic = FALSE,
  keep_full_name = FALSE,
  keep_prefix = TRUE,
  plot_x_size = 9,
  mylabels_x = NULL,
  font_family = NULL,
  ...
)
```

Arguments:

`color_vector` color pallete.

`pheatmap` default FALSE; whether use heatmap with clustering plot.

`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.

`plot_x_size` default 9; x axis text size.

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

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

`...` 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 env_data inside.

Usage:

```
trans_env$plot_scatterfit(
  x = NULL,
  y = NULL,
  use_cor = TRUE,
  cor_method = "pearson",
  add_line = TRUE,
  use_se = TRUE,
  text_x_pos = NULL,
  text_y_pos = NULL,
  x_axis_title = "",
  y_axis_title = "",
  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 env_data 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 env_data inside. If y is a distance matrix, it will be transformed to be a vector.

`use_cor` default TRUE; TRUE for correlation; FALSE for regression.

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

`add_line` default TRUE; whether add the fitted line in the plot.

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

`text_x_pos` default NULL; the central x axis position of the fitting text.

`text_y_pos` default NULL; the central y axis position of the fitting text.

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

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

`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.

`...` the parameters passing to ggplot2::geom_point function.

Returns: plot.

Examples:

```
\donttest{
t1$plot_scatterfit(x = 1, y = 2, alpha = .5)
}
```

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)

## ------------------------------------------------
## Method `trans_env$cal_rda`
## ------------------------------------------------


t1$cal_rda(use_dbrda = TRUE, use_measure = "bray")


## ------------------------------------------------
## Method `trans_env$cal_rda_envsquare`
## ------------------------------------------------


t1$cal_rda_envsquare()


## ------------------------------------------------
## Method `trans_env$trans_rda`
## ------------------------------------------------


t1$trans_rda(adjust_arrow_length = TRUE, max_perc_env = 10)


## ------------------------------------------------
## Method `trans_env$plot_rda`
## ------------------------------------------------


t1$plot_rda(plot_color = "Group")


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


t1$cal_mantel(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_rf$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, alpha = .5)
```

`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 OTU by matching the taxonomic assignments to the functional database; Prokaryotes, based on the FAPROTAX database or NJC19 database, please also cite: FAPROTAX: Louca et al. (2016). Decoupling function and taxonomy in the global ocean microbiome. Science, 353(6305), 1272. <doi:10.1126/science.aaf4507>; 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, based on the FUNGuild database or FungalTraits database, please also cite: 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>

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", selecting a prokaryotic trait database; see the description in this function.

`fungi_database` default "FUNGuild"; "FUNGuild" or "FungalTraits", a fungi trait database for the identification; see the description in this function.

Returns: res_spe_func in object.

Examples:

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

Method `cal_spe_func_perc()`**:** Calculating the percentages of species with specific trait in communities or modules. The percentages of the OTUs with specific trait can reflect the potential of the corresponding function in the community or the module in the network.

Usage:

```
trans_func$cal_spe_func_perc(
  use_community = TRUE,
  abundance_weighted = FALSE,
  node_type_table = NULL
)
```

Arguments:

`use_community` default TRUE; whether calculate community; if FALSE, use module.

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

`node_type_table` default NULL; If use_community FALSE; provide the node_type_table with the module information, such as the result of cal_node_type.

Returns: res_spe_func_perc in object.

Examples:

```
\donttest{
t1$cal_spe_func_perc(use_community = 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 modules.

Usage:

```
trans_func$plot_spe_func_perc(
  filter_func = NULL,
  use_group_list = TRUE,
  add_facet = TRUE,
  select_samples = NULL
)
```

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 use personalized group list, 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

Returns: ggplot2.

Examples:

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

Method `cal_tax4fun()`**:** Predict functional potential of communities using tax4fun. please also 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 otu 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. 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(
  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-2.11.0+/bin ; blast tools folder downloaded from "ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+" ; e.g. ncbi-blast-2.11.0+-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/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.

`database_mode` default 'Ref99NR'; "Ref99NR" or "Ref100NR" .

`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; UProC was used to functionally anotate the genomes in the reference data.

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

`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-2.11.0+/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()


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


t1$cal_spe_func_perc(use_community = 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-2.11.0+/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 related methods, including the correlation based <doi:10.1186/1471-2105-13-113>, SpiecEasi <doi:10.1371/journal.pcbi.1004226>, and Probabilistic Graphical Models based <doi:10.1016/j.cels.2019.08.002> network construction approaches, network and node attributes analysis eigengene analysis, network subsetting 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$cal_eigen()`
- `trans_network$plot_taxa_roles()`
- `trans_network$subset_network()`
- `trans_network$get_edge_table()`
- `trans_network$cal_powerlaw_p()`
- `trans_network$cal_powerlaw_fit()`
- `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 represent do not calculate correlations, used for non-correlation based network.

`taxa_level` default "OTU"; taxonomic rank.

`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; number or name vector to select the physicochemical data in dataset$sample_table.

`add_data` default NULL; provide physicochemical table additionally.

Returns: res_cor_p list.

Examples:

```
\donttest{
data(dataset)
# correlation network
t1 <- trans_network$new(
dataset = dataset,
cal_cor = "base",
taxa_level = "OTU",
filter_thres = 0.001)
}
```

Method `cal_network()`: Calculate network either based on the correlation method or based on SpiecEasi or based on the Probabilistic Graphical Models (PGM) in julia FlashWeave; See Deng et al. (2012) <doi:10.1186/1471-2105-13-113> for correlation based method, Kurtz et al. (2015) <doi:doi:10.1371/journal.pcbi.1004226> for SpiecEasi method, Tackmann et al. (2019) <doi:10.1016/j.cels.2019.08.002> for PGM based method.

Usage:

```
trans_network$cal_network(
  network_method = c("COR", "SpiecEasi", "PGM")[1],
  p_thres = 0.01,
  COR_weight = TRUE,
  COR_p_adjust = "fdr",
  COR_cut = 0.6,
  COR_low_threshold = 0.4,
  COR_optimization = FALSE,
  PGM_meta_data = FALSE,
  PGM_sensitive = "true",
  PGM_heterogeneous = "true",
  SpiecEasi_method = "mb",
  add_taxa_name = "Phylum",
  usename_rawtaxa_when_taxalevel_notOTU = FALSE,
  ...
)
```

Arguments:

`network_method` default "COR"; "COR", "SpiecEasi" or "PGM"; COR: correlation based method; PGM: Probabilistic Graphical Models based method.

`p_thres` default .01; the p value threshold.

`COR_weight` default TRUE; whether use correlation coefficient as the weight of edges.

`COR_p_adjust` default "fdr"; p.adjust method, see p.adjust.methods.

`COR_cut` default .6; correlation coefficient threshold.

`COR_low_threshold` default .4; the lowest correlation coefficient threshold, use with COR_optimization = TRUE.

`COR_optimization` default FALSE; whether use random matrix theory to optimize the choice of correlation coefficient, see https://doi.org/10.1186/1471-2105-13-113

`PGM_meta_data` default FALSE; whether use env data for the optimization, If TRUE, will automatically find the env_data in the object.

`PGM_sensitive` default "true"; whether use sensitive type in the PGM model.

`PGM_heterogeneous` default "true"; whether use heterogeneous type in the PGM model.

`SpiecEasi_method` default "mb"; either 'glasso' or 'mb';see spiec.easi in package SpiecEasi and https://github.com/zdk123/SpiecEasi.

`add_taxa_name` default "Phylum"; NULL or a taxonomic rank name; used to add taxonomic rank name to network.

`usename_rawtaxa_when_taxalevel_notOTU` default FALSE; whether replace the name of nodes using the taxonomic information.

`...` paremeters pass to spiec.easi in package SpiecEasi for network_method = "SpiecEasi".

Returns: res_network in object.

Examples:

```
\donttest{
t1$cal_network(p_thres = 0.01, COR_cut = 0.6)
}
```

Method `cal_module()`**:** Add network modules to the network.

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".

`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: a network with modules, stored in object.

Examples:

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

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.

Returns: None.

Method `cal_network_attr()`**:** Calculate network properties.

Usage:

```
trans_network$cal_network_attr()
```

Returns: res_network_attr in object.

Examples:

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

Method `cal_node_type()`**:** Calculate node properties.

Usage:

```
trans_network$cal_node_type()
```

Returns: res_node_type in object.

Examples:

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

Method `cal_eigen()`**:** Calculate eigengenes of modules, i.e. the first principal component based on PCA analysis, and the percentage of variance.

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_type for the variable names used in the parameters.

Usage:

```
trans_network$plot_taxa_roles(
  use_type = c(1, 2)[1],
  roles_colors = NULL,
  plot_module = FALSE,
  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_colors` default NULL; for use_type 1; colors for each group.

`plot_module` default FALSE; for use_type 1; whether plot the modules information.

`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.

Returns: ggplot.

Examples:

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

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_type %>% .[.$module == "M1", ] %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1
}
```

Method `get_edge_table()`**:** Get the table of edges, including connected nodes, labels and weight.

Usage:

```
trans_network$get_edge_table()
```

Returns: data.frame

Examples:

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

Method `cal_powerlaw_p()`**:** Perform a bootstrapping hypothesis test to determine whether degrees follows a power law distribution; a significant p represents the distribution does not follow power law.

Usage:

```
trans_network$cal_powerlaw_p(...)
```

Arguments:

`...` paremeters pass to bootstrap_p in poweRlaw package.

Returns: two lists stored in object; see estimate_xmin and bootstrap_p in poweRlaw package for the details.

Examples:

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

Method `cal_powerlaw_fit()`**:** Fit degrees to a power law distribution.

Usage:

```
trans_network$cal_powerlaw_fit(xmin = NULL, ...)
```

Arguments:

`xmin` default NULL; See xmin in fit_power_law function; suggest using the result res_powerlaw_min from cal_powerlaw_p function.

`...` paremeters pass to fit_power_law function in igraph package.

Returns: list stored in object; see fit_power_law function for the details explanation.

Examples:

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

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)
# correlation network
t1 <- trans_network$new(
dataset = dataset,
cal_cor = "base",
taxa_level = "OTU",
filter_thres = 0.001)


## ------------------------------------------------
## Method `trans_network$cal_network`
## ------------------------------------------------


t1$cal_network(p_thres = 0.01, COR_cut = 0.6)


## ------------------------------------------------
## Method `trans_network$cal_module`
## ------------------------------------------------


t1$cal_module()


## ------------------------------------------------
## Method `trans_network$cal_network_attr`
## ------------------------------------------------
```

```
t1$cal_network_attr()


## ------------------------------------------------
## Method `trans_network$cal_node_type`
## ------------------------------------------------


t1$cal_node_type()


## ------------------------------------------------
## Method `trans_network$cal_eigen`
## ------------------------------------------------


t1$cal_eigen()


## ------------------------------------------------
## Method `trans_network$plot_taxa_roles`
## ------------------------------------------------


t1$plot_taxa_roles()


## ------------------------------------------------
## Method `trans_network$subset_network`
## ------------------------------------------------


t1$subset_network(node = t1$res_node_type %>% .[.$module == "M1", ] %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1


## ------------------------------------------------
## Method `trans_network$get_edge_table`
## ------------------------------------------------


t1$get_edge_table()


## ------------------------------------------------
## Method `trans_network$cal_powerlaw_p`
## ------------------------------------------------


t1$cal_powerlaw_p()
```

```
## ------------------------------------------------
## Method `trans_network$cal_powerlaw_fit`
## ------------------------------------------------


t1$cal_powerlaw_fit()
```

`trans_nullmodel` *Create trans_nullmodel object.*

Description

This class is a wrapper for a series of null model and phylogeny related approaches, including the mantel correlogram analysis of phylogenetic signal, betaNTI, betaNRI 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>.

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_Cscore()`
- `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 you want to use, if set, filter_thres parameter 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.

Returns: intermediate files in object.

Examples:

```
data(dataset)
data(env_data_16S)
t1 <- trans_nullmodel$new(dataset, taxa_number = 100, 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 to reduce dimensionality.

`break.pts` default seq(0, 1, 0.02); see `mantel.correlog`

`cutoff` default FALSE; see cutoff 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()
```

Returns: ggplot.

Examples:

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

Method `cal_betampd()`**:** Calculate betaMPD. Faster than comdist in picante package.

Usage:

```
trans_nullmodel$cal_betampd(abundance.weighted = FALSE)
```

Arguments:

`abundance.weighted` default FALSE; whether use weighted abundance

Returns: res_betampd in object.

Examples:

```
\donttest{
t1$cal_betampd(abundance.weighted=FALSE)
}
```

Method `cal_betamntd()`**:** Calculate betaMNTD. Faster than comdistnt in picante package.

Usage:

```
trans_nullmodel$cal_betamntd(
  abundance.weighted = FALSE,
  exclude.conspecifics = FALSE
)
```

Arguments:

`abundance.weighted` default FALSE; whether use weighted abundance

`exclude.conspecifics` default FALSE; see comdistnt in picante package.

Returns: res_betamntd in object.

Examples:

```
\donttest{
t1$cal_betamntd(abundance.weighted=FALSE)
}
```

Method `cal_ses_betampd()`**:** Calculate ses.betaMPD (betaNRI).

Usage:

```
trans_nullmodel$cal_ses_betampd(
  runs = 1000,
  abundance.weighted = FALSE,
  verbose = TRUE
)
```

Arguments:

`runs` default 1000; simulation runs.

`abundance.weighted` default FALSE; whether use weighted abundance.

`verbose` default TRUE; whether show the calculation process message.

Returns: res_ses_betampd in object.

Examples:

```
\donttest{
t1$cal_ses_betampd(runs = 100, abundance.weighted = FALSE)
}
```

Method `cal_ses_betamntd()`**:** Calculate ses.betaMNTD (betaNTI).

Usage:

```
trans_nullmodel$cal_ses_betamntd(
  runs = 1000,
  abundance.weighted = FALSE,
  exclude.conspecifics = FALSE,
  verbose = TRUE
)
```

Arguments:

`runs` default 1000; simulation runs.

`abundance.weighted` default FALSE; whether use weighted abundance

`exclude.conspecifics` default FALSE; see comdistnt in picante package.

`verbose` default TRUE; whether show the calculation process message.

Returns: res_ses_betamntd in object.

Examples:

```
\donttest{
t1$cal_ses_betamntd(runs = 100, abundance.weighted = FALSE, exclude.conspecifics = FALSE)
}
```

Method `cal_rcbray()`**:** Calculate 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{
t1$cal_rcbray(runs=200)
}
```

Method `cal_process()`**:** Infer the 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_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 `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, taxa_number = 100, 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=FALSE)

## ------------------------------------------------
## Method `trans_nullmodel$cal_betamntd`
## ------------------------------------------------

t1$cal_betamntd(abundance.weighted=FALSE)

## ------------------------------------------------
## Method `trans_nullmodel$cal_ses_betampd`
## ------------------------------------------------

t1$cal_ses_betampd(runs = 100, abundance.weighted = FALSE)

## ------------------------------------------------
## Method `trans_nullmodel$cal_ses_betamntd`
## ------------------------------------------------

t1$cal_ses_betamntd(runs = 100, abundance.weighted = FALSE, exclude.conspecifics = FALSE)

## ------------------------------------------------
## Method `trans_nullmodel$cal_rcbray`
## ------------------------------------------------

t1$cal_rcbray(runs=200)

## ------------------------------------------------
## Method `trans_nullmodel$cal_process`
## ------------------------------------------------

t1$cal_process(use_betamntd = TRUE)
```

```
## ------------------------------------------------
## Method `trans_nullmodel$cal_Cscore`
## ------------------------------------------------


t1$cal_Cscore()
```

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$trans_venn_com()`
- `trans_venn$print()`
- `trans_venn$clone()`

Method `new()`**:**

Usage:

```
trans_venn$new(dataset = NULL, sample_names = NULL, ratio = "numratio")
```

Arguments:

`dataset` the object of `microtable` Class.

`sample_names` default NULL; if provided, filter the samples.

`ratio` default numratio; NULL, "numratio" or "seqratio"; numratio: calculate number percentage; seqratio: calculate sequence percentage; NULL: no additional percentage.

Returns: venn_table and venn_count_abund 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
)
```

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

Returns: ggplot.

Examples:

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

Method `trans_venn_com()`**:** Transform venn result for the composition analysis.

Usage:

```
trans_venn$trans_venn_com(use_OTUs_frequency = TRUE)
```

Arguments:

`use_OTUs_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_venn_com(use_OTUs_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_venn_com`
## ------------------------------------------------


t2 <- t1$trans_venn_com(use_OTUs_frequency = TRUE)
```

Index

∗ **R6**
  dataset, 3
∗ **data.frame**
  env_data_16S, 4
  fungi_func_FungalTraits, 5
  fungi_func_FUNGuild, 5
  ko_map, 5
  otu_table_16S, 15
  otu_table_ITS, 15
  phylo_tree_16S, 16
  prok_func_FAPROTAX, 16
  prok_func_NJC19_list, 16
  sample_info_16S, 17
  sample_info_ITS, 17
  taxonomy_table_16S, 17
  taxonomy_table_ITS, 18
∗ **list**
  rep_fasta_16S, 16
  Tax4Fun2_KEGG, 17
∗ **object**
  dataset, 3

adonis, *31*
aov, *26*

betadisper, *31*

clone, 2

data.frame, *18*
dataset, 3
dropallfactors, 4

env_data_16S, 4

fungi_func_FungalTraits, 5
fungi_func_FUNGuild, 5

geom_bar, *21*, *37*
geom_boxplot, *23*

ko_map, 5

mantel, *43*
mantel.correlog, *63*
microtable, 6, *25*, *29*, *35*, *40*, *49*, *55*, *63*, *68*, *70*

otu_table_16S, 15
otu_table_ITS, 15

phylo_tree_16S, 16
prok_func_FAPROTAX, 16
prok_func_NJC19_list, 16

rep_fasta_16S, 16
rrarefy, *8*

sample, *8*
sample_info_16S, 17
sample_info_ITS, 17
stat_ellipse, *30*

Tax4Fun2_KEGG, 17
taxonomy_table_16S, 17
taxonomy_table_ITS, 18
tidy_taxonomy, 18
trans_abund, 19
trans_alpha, 25
trans_beta, 28
trans_diff, 34
trans_env, 39
trans_func, 48
trans_network, 54
trans_nullmodel, 62
trans_venn, 68

vegdist, *12*