Package 'microeco'

May 18, 2025

Type Package

Title Microbial Community Ecology Data Analysis

Version 1.15.0

Author Chi Liu [aut, cre],
Felipe R. P. Mansoldo [ctb],
Minjie Yao [ctb],
Xiangzhen Li [ctb]

Maintainer Chi Liu <liuchi0426@126.com>

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

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

Depends R (>= 3.5.0)

Imports R6, stats, ape, vegan, rlang, data.table, magrittr, dplyr, tibble, scales, grid, ggplot2 (>= 3.5.0), RColorBrewer, reshape2, igraph (>= 2.0.0), lifecycle

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

License GPL-3

LazyData true

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2025-05-18 11:10:02 UTC

RoxygenNote 7.3.2

Contents

clone . . . 2
dataset . . . 3
dropallfactors . . . 4
env_data_16S . . . 5
fungi_func_FungalTraits . . . 5
fungi_func_FUNGuild . . . 5
microeco . . . 6
microtable . . . 6
otu_table_16S . . . 18
otu_table_ITS . . . 19
phylo_tree_16S . . . 19
prok_func_FAPROTAX . . . 19
prok_func_NJC19_list . . . 20
sample_info_16S . . . 20
sample_info_ITS . . . 20
Tax4Fun2_KEGG . . . 20
taxonomy_table_16S . . . 21
taxonomy_table_ITS . . . 21
tidy_taxonomy . . . 21
trans_abund . . . 22
trans_alpha . . . 33
trans_beta . . . 39
trans_classifier . . . 49
trans_diff . . . 60
trans_env . . . 70
trans_func . . . 86
trans_network . . . 92
trans_norm . . . 108
trans_nullmodel . . . 112
trans_venn . . . 122

Index **128**

`clone` *Copy an R6 class object*

Description

Copy an R6 class object

Usage

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

Arguments

x	R6 class object
deep	default TRUE; TRUE means deep copy, i.e. copied object is unlinked with the original one.

Value

identical but unlinked R6 object

Examples

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

---

dataset *The dataset structured with microtable class for the demonstration of examples*

Description

The dataset arose from 16S rRNA gene amplicon sequencing of wetland soils in China <doi:10.1016/j.geoderma.2018.09.035>. In `dataset$sample_table`, the 'Group' column means Chinese inland wetlands (IW), coastal wetland (CW) and Tibet plateau wetlands (TW). The column 'Type' denotes the sampling region: northeastern region (NE), northwest region (NW), North China area (NC), middle-lower reaches of the Yangtze River (YML), southern coastal area (SC), upper reaches of the Yangtze River (YU) and Qinghai-Tibet Plateau (QTP). The column 'Saline' represents the saline soils and non-saline soils.

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 object
`unfac2num`	default FALSE; whether try to convert all character columns to numeric directly; If TRUE, it will attempt to convert each column, including those of character and factor types. First, it tries to convert them to the character type, and then checks if they can be converted to numeric. If the conversion to numeric is possible, it outputs the numeric type; otherwise, it outputs the character type. If FALSE, only columns with the factor attribute will be attempted for conversion. Factors will first be converted to character type, and then an attempt will be made to convert them to numeric. If successful, the numeric type will be output; otherwise, the character type will be output. This process can effectively remove the 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. Therefore, this parameter can enforce the conversion of all character and factor types to numeric. This operation is very useful in some cases that numerical data is required as input.

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 example data*

Description

The environmental factors for the 16S example data

Usage

```
data(env_data_16S)
```

fungi_func_FungalTraits *The FungalTraits database for fungi trait prediction*

Description

The FungalTraits database for fungi trait prediction

Usage

```
data(fungi_func_FungalTraits)
```

fungi_func_FUNGuild *The FUNGuild database for fungi trait prediction*

Description

The FUNGuild database for fungi trait prediction

Usage

```
data(fungi_func_FUNGuild)
```

`microeco` *Introduction to microeco package (*Rhref*https://github.com/ChiLiubio/microecohttps://github.com/ChiLiubio/microeco)*

Description

For the detailed tutorial on microeco package, please follow the links:
Online tutorial website: https://chiliubio.github.io/microeco_tutorial/
Download tutorial: https://github.com/ChiLiubio/microeco_tutorial/releases

For each R6 class, please open the help document by searching the class name. For example, to search microtable class, please run the command `help(microtable)` or `?microtable`.
Another way to open the help document of R6 class is to click the following links collected:
`microtable`
`trans_abund`
`trans_venn`
`trans_alpha`
`trans_beta`
`trans_diff`
`trans_network`
`trans_nullmodel`
`trans_classifier`
`trans_env`
`trans_func`
`trans_norm`

To report bugs or discuss questions, please use Github Issues (https://github.com/ChiLiubio/microeco/issues).
Before creating a new issue, please read the guideline (https://chiliubio.github.io/microeco_tutorial/notes.html#github-issues).

To cite microeco package in publications, please run the following command to get the reference:
`citation("microeco")`

Reference:
Chi Liu, Yaoming Cui, Xiangzhen Li and Minjie Yao. 2021. microeco: an R package for data mining in microbial community ecology. FEMS Microbiology Ecology, 97(2): fiaa255. DOI:10.1093/femsec/fiaa255

`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 basic data manipulations, including microtable object creation, data trimming, data filtering, rarefaction based on Paul et al. (2013) <doi:10.1371/journal.pone.0061217>, taxonomic 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.

Online tutorial: https://chiliubio.github.io/microeco_tutorial/
Download tutorial: https://github.com/ChiLiubio/microeco_tutorial/releases

Format

microtable.

Methods

Public methods:

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

Method `new()`:

Usage:

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

Arguments:

`otu_table` data.frame class; The feature abundance table; rownames are features (e.g. OTUs/ASVs/species/genes); column names are samples.

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

`tax_table` default NULL; data.frame class; The taxonomic information table; rownames are features; column names are taxonomic classes.

`phylo_tree` default NULL; phylo class; The phylogenetic tree that must be read with the `read.tree` function of ape package.

`rep_fasta` default NULL; `DNAStringSet`, `list` or `DNAbin` class; The representative sequences of OTUs/ASVs. The sequences should be read with the `readDNAStringSet` function in `Biostrings` package (DNAStringSet class), `read.fasta` function in `seqinr` package (list class), or `read.FASTA` function in ape package (DNAbin class).

`auto_tidy` default FALSE; Whether tidy the data in the `microtable` object automatically. If TRUE, the function can invoke the `tidy_dataset` function.

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

`sample_table` The sample information table.

`otu_table` The feature table.

`tax_table` The taxonomic table.

`phylo_tree` The phylogenetic tree.

`rep_fasta` The sequences.

`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)
m1 <- microtable$new(otu_table = otu_table_16S)
m1 <- microtable$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
m1$tidy_dataset()
```

Method `filter_pollution()`**:** Filter the features considered pollution in `microtable$tax_table`. This operation will remove any line of the `microtable$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: updated microtable object

Examples:

```
m1$filter_pollution(taxa = c("mitochondria", "chloroplast"))
```

Method `filter_taxa()`**:** Filter the features with low abundance and/or low occurrence frequency for `otu_table` or `taxa_abund` list.

Usage:

```
microtable$filter_taxa(
  rel_abund = 0,
  freq = 1,
  include_lowest = TRUE,
  for_taxa_abund = FALSE
)
```

Arguments:

`rel_abund` default 0; the relative abundance threshold, such as 0.0001.

`freq` default 1; the occurrence frequency threshold. For example, the number 2 represents filtering the feature that occurs less than 2 times. A number smaller than 1 is also allowable. For instance, the number 0.1 represents filtering the feature that occurs in less than 10% samples.

`include_lowest` default TRUE; whether include the feature with the threshold.

`for_taxa_abund` default FALSE; whether apply this function to `taxa_abund` list. FALSE means using this function for `otu_table`

Returns: updated microtable object

Examples:

```
\donttest{
d1 <- clone(m1)
d1$filter_taxa(rel_abund = 0.0001, freq = 0.2)
}
```

Method `rarefy_samples()`**:** Rarefy communities to make all samples have same count number.

Usage:

```
microtable$rarefy_samples(
  method = c("rarefy", "SRS")[1],
  sample.size = NULL,
  ...
)
```

Arguments:

`method` default c("rarefy", "SRS")[1]; "rarefy" represents the classical resampling like `rrarefy` function of `vegan` package. "SRS" is scaling with ranked subsampling method based on the SRS package provided by Lukas Beule and Petr Karlovsky (2020) <DOI:10.7717/peerj.9593>.

`sample.size` default NULL; libray size. If not provided, use the minimum number across all samples. For "SRS" method, this parameter is passed to `Cmin` parameter of `SRS` function of SRS package.

`...` parameters pass to `norm` function of `trans_norm` class.

Returns: rarefied microtable object.

Examples:

```
\donttest{
m1$rarefy_samples(sample.size = min(m1$sample_sums()))
}
```

Method `tidy_dataset()`**:** Trim all the data in the `microtable` object to make taxa and samples consistent. The results are intersections across data.

Usage:

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

Arguments:

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

Returns: None. The data in the object are tidied up. If `tax_table` is in object, its row names are completely same with the row names of `otu_table`.

Examples:

```
m1$tidy_dataset(main_data = TRUE)
```

Method `add_rownames2taxonomy()`**:** Add the row names of `microtable$tax_table` as its last column. This is especially useful when the row names of `microtable$tax_table` are required as a taxonomic level for the taxonomic abundance calculation and biomarker identification.

Usage:

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

Arguments:

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

Returns: tax_table updated in the object.

Examples:

```
\donttest{
m1$add_rownames2taxonomy()
}
```

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

Usage:

```
microtable$sample_sums()
```

Returns: abundance in each sample.

Examples:

```
\donttest{
m1$sample_sums()
}
```

Method `taxa_sums()`**:** Sum the abundance for each taxon.

Usage:

```
microtable$taxa_sums()
```

Returns: abundance in each taxon.

Examples:

```
\donttest{
m1$taxa_sums()
}
```

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

Usage:

```
microtable$sample_names()
```

Returns: sample names.

Examples:

```
\donttest{
m1$sample_names()
}
```

Method `taxa_names()`**:** Show the taxa names.

Usage:

```
microtable$taxa_names()
```

Returns: taxa names.

Examples:

```
\donttest{
m1$taxa_names()
}
```

Method `rename_taxa()`**:** Rename the features, including the row names of `otu_table`, row names of `tax_table`, tip labels of `phylo_tree` and names in `rep_fasta`.

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 order of row names in `otu_table`.

Returns: renamed object

Examples:

```
\donttest{
m1$rename_taxa()
}
```

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

Usage:

```
microtable$merge_samples(group)
```

Arguments:

group a column name in sample_table of microtable object.

Returns: a merged microtable object.

Examples:

```
\donttest{
m1$merge_samples("Group")
}
```

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

Usage:

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

Arguments:

taxa default "Genus"; the specific rank in tax_table.

Returns: a merged `microtable` object.

Examples:

```
\donttest{
m1$merge_taxa(taxa = "Genus")
}
```

Method `save_table()`**:** Save each basic data in microtable object as local file.

Usage:

```
microtable$save_table(dirpath = "basic_files", sep = ",", ...)
```

Arguments:

dirpath default "basic_files"; directory to save the tables, phylogenetic tree and sequences in microtable object. It will be created if not found.

sep default ","; the field separator string, used to save tables. Same with sep parameter in write.table function. default ',' correspond to the file name suffix 'csv'. The option '\t' correspond to the file name suffix 'tsv'. For other options, suffix are all 'txt'.

... parameters passed to write.table.

Examples:

```
\dontrun{
m1$save_table()
}
```

Method `cal_abund()`**:** Calculate the taxonomic abundance at each taxonomic level or selected levels.

Usage:

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

Arguments:

`select_cols` default NULL; numeric vector (column sequences) or character vector (column names of `microtable$tax_table`); applied to select columns to calculate abundances according to ordered hierarchical levels. This parameter is very useful when only part of the columns are needed to calculate abundances.

`rel` default TRUE; if TRUE, relative abundance is used; if FALSE, absolute abundance (i.e. raw values) will be summed.

`merge_by` default "|"; the symbol to merge and concatenate taxonomic names of different levels.

`split_group` default FALSE; if TRUE, split the rows to multiple rows according to one or more columns in `tax_table` when there is multiple mapping information.

`split_by` default "&"; Separator delimiting collapsed values; only available when `split_group = TRUE`.

`split_column` default NULL; one column name used for the splitting in tax_table for each abundance calculation; only available when `split_group = TRUE`. If not provided, the function will split each column that containing the `split_by` character.

`split_special_char` default "&&"; special character that will be used forcibly to split multiple mapping information in `tax_table` by default no matter `split_group` setting.

Returns: `taxa_abund` list in object.

Examples:

```
\donttest{
m1$cal_abund()
}
```

Method `save_abund()`**:** Save taxonomic abundance as local file.

Usage:

```
microtable$save_abund(
  dirpath = "taxa_abund",
  merge_all = FALSE,
  rm_un = FALSE,
  rm_pattern = "__$",
  sep = ",",
  ...
)
```

Arguments:

`dirpath` default "taxa_abund"; directory to save the taxonomic abundance files. It will be created if not found.

`merge_all` default FALSE; Whether merge all tables into one. The merged file format is generally called 'mpa' style.

`rm_un` default FALSE; Whether remove unclassified taxa in which the name ends with '__' generally.

`rm_pattern` default "__$"; The pattern searched through the merged taxonomic names. See also `pattern` parameter in `grepl` function. Only available when `rm_un = TRUE`. The default "__$" means removing the names end with '__'.

`sep` default ","; the field separator string. Same with `sep` parameter in `write.table` function. default `','` correspond to the file name suffix 'csv'. The option `'\t'` correspond to the file name suffix 'tsv'. For other options, suffix are all 'txt'.

`...` parameters passed to `write.table`.

Examples:

```
\dontrun{
m1$save_abund(dirpath = "taxa_abund")
m1$save_abund(merge_all = TRUE, rm_un = TRUE, sep = "\t")
}
```

Method `cal_alphadiv()`**:** Calculate alpha diversity.

Usage:

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

Arguments:

`measures` default NULL; one or more indexes in `c("Observed", "Coverage", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson", "Fisher", "Pielou")`; The default NULL represents that all the measures are calculated. 'Shannon', 'Simpson' and 'InvSimpson' are calculated based on `vegan::diversity` function; 'Chao1' and 'ACE' depend on the function `vegan::estimateR`. 'Fisher' index relies on the function `vegan::fisher.alpha`. "Observed" means the observed species number in a community, i.e. richness. "Coverage" represents good's coverage. It is defined:

$$Coverage = 1 - \frac{f1}{n}$$

where *n* is the total abundance of a sample, and *f1* is the number of singleton (species with abundance 1) in the sample. "Pielou" denotes the Pielou evenness index. It is defined:

$$J = \frac{H'}{\ln(S)}$$

where *H'* is Shannon index, and *S* is the species number.

`PD` default FALSE; whether Faith's phylogenetic diversity is calculated. The calculation depends on the function `picante::pd`. Note that the phylogenetic tree (`phylo_tree` object in the data) is required for PD.

Returns: alpha_diversity stored in the object. The se.chao1 and se.ACE are the standard erros of Chao1 and ACE, respectively.

Examples:

```
\donttest{
m1$cal_alphadiv(measures = NULL, PD = FALSE)
class(m1$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 dissimilarity matrix, such as 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,
  force_jaccard_binary = TRUE,
  ...
)
```

Arguments:

`method` default NULL; a character vector with one or more elements; `c("bray", "jaccard")` is used when `method = NULL`; See the `method` parameter in `vegdist` function for more available options, such as 'aitchison' and 'robust.aitchison'.

`unifrac` default FALSE; whether UniFrac indexes (weighted and unweighted) are calculated. Phylogenetic tree is necessary when `unifrac = TRUE`.

`binary` default FALSE; Whether convert abundance to binary data (presence/absence).

`force_jaccard_binary` default TRUE; Whether forcibly convert abundance to binary data (presence/absence) when `method = "jaccard"`. The reason for this setting is that the Jaccard metric is commonly used for binary data. If `force_jaccard_binary = FALSE` is set, the conversion will not be enforced, but will instead be based on the setting of the `binary` parameter.

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

Returns: beta_diversity list stored in the object.

Examples:

```
\donttest{
m1$cal_betadiv(unifrac = FALSE)
class(m1$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 `print()`: Print the microtable object.

Usage:

```
microtable$print()
```

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)
m1 <- microtable$new(otu_table = otu_table_16S)
m1 <- microtable$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
m1$tidy_dataset()

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

m1$filter_pollution(taxa = c("mitochondria", "chloroplast"))

## ------------------------------------------------
## Method `microtable$filter_taxa`
## ------------------------------------------------


d1 <- clone(m1)
d1$filter_taxa(rel_abund = 0.0001, freq = 0.2)


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


m1$rarefy_samples(sample.size = min(m1$sample_sums()))


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

m1$tidy_dataset(main_data = TRUE)

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


m1$add_rownames2taxonomy()
```

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


m1$sample_sums()


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


m1$taxa_sums()


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


m1$sample_names()


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


m1$taxa_names()


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


m1$rename_taxa()


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


m1$merge_samples("Group")


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

```
m1$merge_taxa(taxa = "Genus")


## ------------------------------------------------
## Method `microtable$save_table`
## ------------------------------------------------

## Not run:
m1$save_table()

## End(Not run)

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


m1$cal_abund()


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

## Not run:
m1$save_abund(dirpath = "taxa_abund")
m1$save_abund(merge_all = TRUE, rm_un = TRUE, sep = "\t")

## End(Not run)

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


m1$cal_alphadiv(measures = NULL, PD = FALSE)
class(m1$alpha_diversity)


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


m1$cal_betadiv(unifrac = FALSE)
class(m1$beta_diversity)
```

---

otu_table_16S *The OTU table of the 16S example data*

Description

The OTU table of the 16S example data

Usage

```
data(otu_table_16S)
```

`otu_table_ITS` *The OTU table of the ITS example data*

Description

The OTU table of the ITS example data

Usage

```
data(otu_table_ITS)
```

`phylo_tree_16S` *The phylogenetic tree of 16S example data*

Description

The phylogenetic tree of 16S example data

Usage

```
data(phylo_tree_16S)
```

`prok_func_FAPROTAX` *The modified FAPROTAX trait database*

Description

The modified FAPROTAX trait database

Usage

```
data(prok_func_FAPROTAX)
```

prok_func_NJC19_list *The modified NJC19 database*

Description

The modified NJC19 database

Usage

```
data(prok_func_NJC19_list)
```

sample_info_16S *The sample information of 16S example data*

Description

The sample information of 16S example data

Usage

```
data(sample_info_16S)
```

sample_info_ITS *The sample information of ITS example data*

Description

The sample information of ITS example data

Usage

```
data(sample_info_ITS)
```

Tax4Fun2_KEGG *The KEGG data files used in the trans_func class*

Description

The KEGG data files used in the trans_func class

Usage

```
data(Tax4Fun2_KEGG)
```

taxonomy_table_16S *The taxonomic information of 16S example data*

Description

The taxonomic information of 16S example data

Usage

```
data(taxonomy_table_16S)
```

taxonomy_table_ITS *The taxonomic information of ITS example data*

Description

The taxonomic information of ITS example data

Usage

```
data(taxonomy_table_ITS)
```

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

Description

Clean up the taxonomic table to make taxonomic assignments consistent.

Usage

```
tidy_taxonomy(
  taxonomy_table,
  column = "all",
  pattern = c(".*unassigned.*", ".*uncultur.*", ".*unknown.*", ".*unidentif.*",
    ".*unclassified.*", ".*No blast hit.*", ".*Incertae.sedis.*"),
  replacement = "",
  ignore.case = TRUE,
  na_fill = ""
)
```

Arguments

taxonomy_table	a data.frame with taxonomic information (rows are features; columns are taxonomic levels); or a microtable object with tax_table in it.
column	default "all"; "all" or a number; 'all' represents cleaning up all the columns; a number represents cleaning up this specific column.
pattern	default c(".*unassigned.*", ".*uncultur.*", ".*unknown.*", ".*unidentif.*", ".*unclassified.*", ".*No blast hit.*", ".*Incertae.sedis.*"); the characters (regular expressions) to be removed or replaced; removed when parameter replacement = "", replaced when parameter replacement has something; Note that the capital and small letters are not distinguished when ignore.case = TRUE.
replacement	default ""; the characters used to replace the character in pattern parameter.
ignore.case	default TRUE; if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.
na_fill	default ""; used to replace NA.

Format

data.frame object.

Value

data.frame

Examples

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

---

trans_abund *Create* trans_abund *object for taxonomic abundance visualization.*

Description

This class is a wrapper for the taxonomic abundance transformations and visualization (e.g., bar plot, boxplot, heatmap, pie chart and line chart). The converted data style is the long-format for ggplot2 plot.

Methods

Public methods:

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

- trans_abund$plot_pie()
- trans_abund$plot_donut()
- trans_abund$plot_radar()
- trans_abund$plot_tern()
- trans_abund$print()
- trans_abund$clone()

Method `new()`**:**

Usage:

```
trans_abund$new(
  dataset = NULL,
  taxrank = "Phylum",
  show = 0,
  ntaxa = 10,
  groupmean = NULL,
  group_morestats = FALSE,
  delete_taxonomy_lineage = TRUE,
  delete_taxonomy_prefix = TRUE,
  prefix = NULL,
  use_percentage = TRUE,
  input_taxaname = NULL,
  high_level = NULL,
  high_level_fix_nsub = NULL
)
```

Arguments:

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

`taxrank` default "Phylum"; taxonomic level, i.e. a column name in `tax_table` of the input object. The function extracts the abundance from the `taxa_abund` list according to the names in the list. If the `taxa_abund` list is NULL, the function can automatically calculate the relative abundance to generate `taxa_abund` list.

`show` default 0; the mean relative abundance threshold for filtering the taxa with low abundance.

`ntaxa` default 10; how many taxa are selected to use. Taxa are ordered by abundance from high to low. This parameter does not conflict with the parameter `show`. Both can be used. `ntaxa = NULL` means the parameter will be invalid.

`groupmean` default NULL; calculate mean abundance for each group. Select a column name in `microtable$sample_table`.

`group_morestats` default FALSE; only available when `groupmean` parameter is provided; Whether output more statistics for each group, including min, max, median and quantile; Thereinto, quantile25 and quantile75 denote 25% and 75% quantiles, respectively.

`delete_taxonomy_lineage` default TRUE; whether delete the taxonomy lineage in front of the target level.

`delete_taxonomy_prefix` default TRUE; whether delete the prefix of taxonomy, such as "g__".

`prefix` default NULL; character string; available when `delete_taxonomy_prefix = T`; default NULL represents using the "letter+__", e.g. "k__" for Phylum level; Please provide the customized prefix when it is not standard, otherwise the program can not correctly recognize it.

use_percentage default TRUE; whether show the abundance percentage. If TRUE, the abundance data will be multiplied by 100.

input_taxaname default NULL; character vector; input taxa names to select some taxa.

high_level default NULL; a taxonomic rank, such as "Phylum", used to add the taxonomic information of higher level. It is required for the legend with nested taxonomic levels in the bar plot or the higher taxonomic level in facets of y axis in the heatmap.

high_level_fix_nsub default NULL; an integer, used to fix the number of selected abundant taxa in each taxon from higher taxonomic level. If the total number under one taxon of higher level is less than the high_level_fix_nsub, the total number will be used. When high_level_fix_nsub is provided, the taxa number of higher level is calculated as: ceiling(ntaxa/high_level_f Note that ntaxa means either the parameter ntaxa or the taxonomic number obtained by filtering according to the show parameter.

Returns: data_abund stored in the object. The column 'all_mean_abund' represents mean relative abundance across all the samples. So the values in one taxon are all same across all the samples. If the sum of column 'Abundance' in one sample is larger than 1, the 'Abundance', 'SD' and 'SE' has been multiplied by 100.

Examples:

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

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

Usage:

```
trans_abund$plot_bar(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  bar_full = TRUE,
  others_color = "grey90",
  facet = NULL,
  order_x = NULL,
  x_axis_name = NULL,
  barwidth = NULL,
  use_alluvium = FALSE,
  clustering = FALSE,
  clustering_plot = FALSE,
  cluster_plot_width = 0.2,
  facet_color = "grey95",
  strip_text = 11,
  legend_text_italic = FALSE,
  xtext_angle = 0,
  xtext_size = 10,
  xtext_keep = TRUE,
  xtitle_keep = TRUE,
  ytitle_size = 17,
  coord_flip = FALSE,
  ggnested = FALSE,
  high_level_add_other = FALSE,
```

```
  bar_type = deprecated()
)
```

Arguments:

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

`bar_full` default TRUE; Whether the bar shows all the features (including 'Others'). Default `TRUE` means total abundance are summed to 1 or 100 (percentage). `FALSE` means 'Others' will not be shown.

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

`facet` default NULL; a character vector for the facet; group column name of `sample_table`, such as, `"Group"`; If multiple facets are needed, please provide ordered names, such as `c("Group", "Type")`. The latter should have a finer scale than the former one; Please adjust the facet orders in the plot by assigning factors in `sample_table` before creating `trans_abund` object or assigning factors in the `data_abund` table of `trans_abund` object. When multiple facets are used, please first install package `ggh4x` using the command `install.packages("ggh4x")`.

`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")`.

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

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

`use_alluvium` default FALSE; whether add alluvium plot. If `TRUE`, please first install `ggalluvial` package.

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

`clustering_plot` default FALSE; whether add clustering plot. If `clustering_plot = TRUE`, `clustering` will be also TRUE in any case for the clustering.

`cluster_plot_width` default 0.2, the dendrogram plot width; available when `clustering_plot = TRUE`.

`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_angle` default 0; number ranging from 0 to 90; used to adjust x axis text angle to reduce text overlap;

`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.

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

`ggnested` default FALSE; whether use nested legend. Need `ggnested` package to be installed (https://github.com/gmteunisse/ggnested). To make it available, please assign `high_level` parameter when creating the object.

`high_level_add_other` default FALSE; whether add 'Others' (all the unknown taxa) in each taxon of higher taxonomic level. Only available when `ggnested = TRUE`.

`bar_type` deprecated. Please use `bar_full` argument instead.

Returns: ggplot2 object.

Examples:

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

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

Usage:

```
trans_abund$plot_heatmap(
  color_values = rev(RColorBrewer::brewer.pal(n = 11, name = "RdYlBu")),
  facet = NULL,
  facet_switch = "y",
  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_keep = TRUE,
  xtext_angle = 0,
  xtext_size = 10,
  ytext_size = 11,
  xtitle_keep = TRUE,
  grid_clean = TRUE,
  legend_title = "% Relative\nAbundance",
  pheatmap = FALSE,
  ...
)
```

Arguments:

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

`facet` default NULL; a character vector for the facet; a group column name of `sample_table`, such as, `"Group"`; If multiple facets are needed, please provide ordered names, such as `c("Group", "Type")`. The latter should have a finer scale than the former one; Please adjust the facet orders in the plot by assigning factors in `sample_table` before creating `trans_abund` object or assigning factors in the `data_abund` table of `trans_abund` object. When multiple facets are used, please first install package `ggh4x` using the command `install.packages("ggh4x")`.

`facet_switch` default "y"; By default, the labels in facets are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both". When the `high_level` is found in the object, the function will generate facets for the higher taxonomy in y axis.

So the default "y" of the parameter is to make the visualization better when `high_level` is found. This parameter will be passed to the `switch` parameter in `ggplot2::facet_grid` or `ggh4x::facet_nested` function.

`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_keep` default TRUE; whether retain x text.

`xtext_angle` default 0; number ranging from 0 to 90; used to adjust x axis text angle to reduce text overlap;

`xtext_size` default 10; x axis text size.

`ytext_size` default 11; y axis text size.

`xtitle_keep` default TRUE; whether retain x title.

`grid_clean` default TRUE; whether remove grid lines.

`legend_title` default "% Relative\nAbundance"; legend title text.

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

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

Returns: ggplot2 object or grid object based on pheatmap.

Examples:

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

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

Usage:

```
trans_abund$plot_box(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  group = NULL,
  show_point = FALSE,
  point_color = "black",
  point_size = 3,
  point_alpha = 0.3,
  plot_flip = FALSE,
```

```
  boxfill = TRUE,
  middlecolor = "grey95",
  middlesize = 1,
  xtext_angle = 0,
  xtext_size = 10,
  ytitle_size = 17,
  ...
)
```

Arguments:

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

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

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

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

`point_size` default 3; If show_point TRUE; use the size

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

`plot_flip` default FALSE; Whether rotate plot.

`boxfill` default TRUE; Whether fill the box with colors.

`middlecolor` default "grey95"; The middle line color.

`middlesize` default 1; The middle line size.

`xtext_angle` default 0; number ranging from 0 to 90; used to adjust x axis text angle to reduce text overlap;

`xtext_size` default 10; x axis text size.

`ytitle_size` default 17; y axis title size.

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

Returns: ggplot2 object.

Examples:

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

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

Usage:

```
trans_abund$plot_line(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  plot_SE = TRUE,
  position = position_dodge(0.1),
  errorbar_size = 1,
  errorbar_width = 0.1,
  point_size = 3,
  point_alpha = 0.8,
  line_size = 0.8,
  line_alpha = 0.8,
  line_type = 1,
  xtext_angle = 0,
  xtext_size = 10,
```

```
  ytitle_size = 17
)
```

Arguments:

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

`plot_SE` default TRUE; TRUE: the errorbar is $meanse$; FALSE: the errorbar is $meansd$.

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

`errorbar_size` default 1; errorbar line size.

`errorbar_width` default 0.1; errorbar width.

`point_size` default 3; point size for taxa.

`point_alpha` default 0.8; point transparency.

`line_size` default 0.8; line size.

`line_alpha` default 0.8; line transparency.

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

`xtext_angle` default 0; number ranging from 0 to 90; used to adjust x axis text angle to reduce text overlap;

`xtext_size` default 10; x axis text size.

`ytitle_size` default 17; y axis title size.

Returns: ggplot2 object.

Examples:

```
\donttest{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5)
t1$plot_line(point_size = 3)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5, groupmean = "Group")
t1$plot_line(point_size = 5, errorbar_size = 1, xtext_angle = 30)
}
```

Method `plot_pie()`**:** Pie chart.

Usage:

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

Arguments:

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

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

`strip_text` default 11; sample title size.

`add_label` default FALSE; Whether add the percentage label in each section of pie chart.

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

Returns: ggplot2 object.

Examples:

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

Method `plot_donut()`**:** Donut chart based on the `ggpubr::ggdonutchart` function.

Usage:

```
trans_abund$plot_donut(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  label = TRUE,
  facet_nrow = 1,
  legend_text_italic = FALSE,
  ...
)
```

Arguments:

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

`label` default TRUE; whether show the percentage label.

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

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

`...` parameters passed to `ggpubr::ggdonutchart`.

Returns: combined ggplot2 objects list, generated by `ggpubr::ggarrange` function.

Examples:

```
\dontrun{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_donut(label = TRUE)
}
```

Method `plot_radar()`**:** Radar chart based on the `ggradar` package (https://github.com/ricardo-bion/ggradar).

Usage:

```
trans_abund$plot_radar(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  ...
)
```

Arguments:

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

`...` parameters passed to `ggradar::ggradar` function except group.colours parameter.

Returns: ggplot2 object.

Examples:

```
\dontrun{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_radar()
}
```

Method `plot_tern()`**:** Ternary diagrams based on the `ggtern` package.

Usage:

```
trans_abund$plot_tern(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  color_legend_guide_size = 4
)
```

Arguments:

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

`color_legend_guide_size` default 4; The size of legend guide for color.

Returns: ggplot2 object.

Examples:

```
\dontrun{
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_tern()
}
```

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

Usage:

```
trans_abund$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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


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


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


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


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

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


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


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


t1$plot_box(group = "Group")


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


t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5)
t1$plot_line(point_size = 3)
t1 <- trans_abund$new(dataset = dataset, taxrank = "Genus", ntaxa = 5, groupmean = "Group")
t1$plot_line(point_size = 5, errorbar_size = 1, xtext_angle = 30)


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


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


## ------------------------------------------------
## Method `trans_abund$plot_donut`
## ------------------------------------------------

## Not run:
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_donut(label = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `trans_abund$plot_radar`
## ------------------------------------------------

## Not run:
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_radar()
```

```
## End(Not run)

## ------------------------------------------------
## Method `trans_abund$plot_tern`
## ------------------------------------------------

## Not run:
t1 <- trans_abund$new(dataset = dataset, taxrank = "Phylum", ntaxa = 6, groupmean = "Group")
t1$plot_tern()

## End(Not run)
```

`trans_alpha` *Create* `trans_alpha` *object for alpha diversity statistics and visualization.*

Description

This class is a wrapper for a series of alpha diversity analysis, including the statistics and visualization.

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,
  by_group = NULL,
  by_ID = NULL,
  order_x = NULL
)
```

Arguments:

`dataset` `microtable` object.

`group` default NULL; a column name of `sample_table` in the input microtable object used for the statistics across groups.

`by_group` default NULL; a column name of `sample_table` used to perform the differential test among groups (from `group` parameter) for each group (from `by_group` parameter) separately.

by_ID default NULL; a column name of sample_table used to perform paired T test or paired Wilcoxon test for the paired data, such as continuous sampling of individual animals or plant compartments for different plant species (ID). So by_ID in sample_table should be the smallest unit of sample collection without any repetition in it. When the by_ID parameter is provided, the function can automatically perform paired test, and no more parameters is required.

order_x default NULL; a column name of sample_table or a vector with sample names. If provided, sort samples using factor.

Returns: data_alpha and data_stat stored in the object.

Examples:

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

Method `cal_diff()`**:** Differential test on alpha diversity.

Usage:

```
trans_alpha$cal_diff(
  measure = NULL,
 method = c("KW", "KW_dunn", "wilcox", "t.test", "anova", "scheirerRayHare", "lm",
    "lme", "betareg", "glmm", "glmm_beta")[1],
  formula = NULL,
  p_adjust_method = "fdr",
  KW_dunn_letter = TRUE,
  alpha = 0.05,
  anova_post_test = "duncan.test",
  anova_varequal_test = FALSE,
  return_model = FALSE,
  ...
)
```

Arguments:

measure default NULL; character vector; If NULL, all indexes will be used; see names of microtable$alpha_diversity, e.g. c("Observed", "Chao1", "Shannon").

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

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

'KW_dunn' Dunn's Kruskal-Wallis Multiple Comparisons <10.1080/00401706.1964.10490181> based on dunnTest function in FSA package

'wilcox' Wilcoxon Rank Sum Test for all paired groups When by_ID parameter is provided in creating the object of the class, paired Wilcoxon test will be performed.

't.test' Student's t-Test for all paired groups. When by_ID parameter is provided in creating the object of the class, paired t-test will be performed.

'anova' Variance analysis. For one-way anova, the default post hoc test is Duncan's new multiple range test. Please use anova_post_test parameter to change the post hoc method. For multi-way anova, Please use formula parameter to specify the model and see aov for more details

'scheirerRayHare' Scheirer-Ray-Hare test (nonparametric test) for a two-way factorial experiment; see `scheirerRayHare` function of `rcompanion` package

'lm' Linear Model based on the `lm` function

'lme' Linear Mixed Effect Model based on the `lmerTest` package

'betareg' Beta Regression for Rates and Proportions based on the `betareg` package

'glmm' Generalized linear mixed model (GLMM) based on the `glmmTMB` package. A family function can be provided using parameter passing, such as: `family = glmmTMB::lognormal(link = "log")`

'glmm_beta' Generalized linear mixed model (GLMM) with a family function of beta distribution. This is an extension of the GLMM model in `'glmm'` option. The only difference is in `glmm_beta` the family function is fixed with the beta distribution function, facilitating the fitting for proportional data (ranging from 0 to 1). The link function is fixed with `"logit"`.

`formula` default NULL; applied to two-way or multi-factor analysis when method is `"anova"`, `"scheirerRayHare"`, `"lm"`, `"lme"`, `"betareg"` or `"glmm"`; specified set for independent variables, i.e. the latter part of a general formula, such as `'block + N*P*K'`.

`p_adjust_method` default "fdr" (for "KW", "wilcox", "t.test" methods) or "holm" (for "KW_dunn"); P value adjustment method; For `method = 'KW', 'wilcox' or 't.test'`, please see `method` parameter of `p.adjust` function for available options; For `method = 'KW_dunn'`, please see `dunn.test::p.adjustment.methods` for available options.

`KW_dunn_letter` default TRUE; For `method = 'KW_dunn'`, `TRUE` denotes significances are presented by letters; `FALSE` means significances are shown by asterisk for paired comparison.

`alpha` default 0.05; Significant level; used for generating significance letters when method is 'anova' or 'KW_dunn'.

`anova_post_test` default "duncan.test". The post hoc test method for one-way anova. The default option represents the Duncan's new multiple range test. Other available options include "LSD.test" (LSD post hoc test) and "HSD.test" (HSD post hoc test). All those are the function names from `agricolae` package.

`anova_varequal_test` default FALSE; whether conduct Levene's Test for equality of variances. Only available for one-way anova. Significant P value means the variance among groups is not equal.

`return_model` default FALSE; whether return the original "lm", "lmer" or "glmm" model list in the object.

`...` parameters passed to `kruskal.test` (when `method = "KW"`) or `wilcox.test` function (when `method = "wilcox"`) or `dunnTest` function of `FSA` package (when `method = "KW_dunn"`) or `agricolae::duncan.test/agricolae::LSD.test/agricolae::HSD.test` (when `method = "anova"`, one-way anova) or `rcompanion::scheirerRayHare` (when `method = "scheirerRayHare"`) or `stats::lm` (when `method = "lm"`) or `lmerTest::lmer` (when `method = "lme"`) or `betareg::betareg` (when `method = "betareg"`) or `glmmTMB::glmmTMB` (when `method = "glmm"`).

Returns: `res_diff`, stored in object with the format `data.frame`.
When method is "betareg", "lm", "lme" or "glmm", "Estimate" and "Std.Error" columns represent the fitted coefficient and its standard error, respectively.

Examples:

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

```
t1 <- trans_alpha$new(dataset = dataset, group = "Type", by_group = "Group")
t1$cal_diff(method = "anova")
}
```

Method `plot_alpha()`**:** Plot the alpha diversity. Box plot (and others for visualizing data in groups of single factor) is used for the visualization of alpha diversity when the `group` is found in the object. When the formula is found in the `res_diff` table in the object, heatmap is employed automatically to show the significances of differential test for multiple indexes, and errorbar (coefficient and standard errors) can be used for single index.

Usage:

```
trans_alpha$plot_alpha(
  plot_type = "ggboxplot",
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  measure = "Shannon",
  group = NULL,
  add = NULL,
  add_sig = TRUE,
  add_sig_label = "Significance",
  add_sig_text_size = 3.88,
  add_sig_label_num_dec = 4,
  order_x_mean = FALSE,
  y_start = 0.1,
  y_increase = 0.05,
  xtext_angle = 30,
  xtext_size = 13,
  ytitle_size = 17,
  bar_width = 0.9,
  bar_alpha = 0.8,
  dodge_width = 0.9,
  plot_SE = TRUE,
  errorbar_size = 1,
  errorbar_width = 0.2,
  errorbar_addpoint = TRUE,
  errorbar_color_black = FALSE,
  point_size = 3,
  point_alpha = 0.8,
  add_line = FALSE,
  line_size = 0.8,
  line_type = 2,
  line_color = "grey50",
  line_alpha = 0.5,
  heatmap_cell = "P.unadj",
  heatmap_sig = "Significance",
  heatmap_x = "Factors",
  heatmap_y = "Measure",
  heatmap_lab_fill = "P value",
  coefplot_sig_pos = 2,
  ...
)
```

Arguments:

`plot_type` default "ggboxplot"; plot type; available options include "ggboxplot", "ggdotplot", "ggviolin", "ggstripchart", "ggerrorplot", "errorbar" and "barerrorbar". The options starting with "gg" are function names coming from `ggpubr` package. All those methods with `ggpubr` package use the `data_alpha` table in the object. "errorbar" represents Mean±SD or Mean±SE plot based on `ggplot2` package by invoking the `data_stat` table in the object. "barerrorbar" denotes "bar plot + error bar". It is similar with "errorbar" and has a bar plot.

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

`measure` default "Shannon"; one alpha diversity index in the object.

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

`add` default NULL; add another plot element; passed to the `add` parameter of the function (e.g., `ggboxplot`) from `ggpubr` package when `plot_type` starts with "gg" (functions coming from ggpubr package).

`add_sig` default TRUE; whether add significance label using the result of `cal_diff` function, i.e. `object$res_diff`; This is manily designed to add post hoc test of anova or other significances to make the label mapping easy.

`add_sig_label` default "Significance"; select a colname of `object$res_diff` for the label text when 'Letter' is not in the table, such as 'P.adj' or 'Significance'.

`add_sig_text_size` default 3.88; the size of text in added label.

`add_sig_label_num_dec` default 4; reserved decimal places when the parameter `add_sig_label` use numeric column, like 'P.adj'.

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

`y_start` default 0.1; the y axis value from which to add the significance asterisk label; the default 0.1 means `max(values) + 0.1 * (max(values) - min(values))`.

`y_increase` default 0.05; the increasing y axia space to add the label (asterisk or letter); the default 0.05 means `0.05 * (max(values) - min(values))`; this parameter is also used to label the letters of anova result with the fixed space.

`xtext_angle` default 30; number (e.g. 30). Angle of text in x axis.

`xtext_size` default 13; x axis text size. NULL means the default size in ggplot2.

`ytitle_size` default 17; y axis title size.

`bar_width` default 0.9; the bar width when `plot_type = "barerrorbar"`.

`bar_alpha` default 0.8; the alpha of bar color when `plot_type = "barerrorbar"`.

`dodge_width` default 0.9; the dodge width used in `position_dodge` function of ggplot2 package when `plot_type` is "errorbar" or "barerrorbar".

`plot_SE` default TRUE; TRUE: the errorbar is $meanse$; FALSE: the errorbar is $meansd$. Available when `plot_type` is "errorbar" or "barerrorbar".

`errorbar_size` default 1; errorbar size. Available when `plot_type` is "errorbar" or "barerrorbar".

`errorbar_width` default 0.2; errorbar width. Available when `plot_type` is "errorbar" or "barerrorbar" and `by_group` is NULL.

`errorbar_addpoint` default TRUE; whether add point for mean. Available when `plot_type` is "errorbar" or "barerrorbar" and `by_group` is NULL.

`errorbar_color_black` default FALSE; whether use black for the color of errorbar when `plot_type` is "errorbar" or "barerrorbar".

point_size default 3; point size for taxa. Available when plot_type is "errorbar" or "barerrorbar".

point_alpha default 0.8; point transparency. Available when plot_type is "errorbar" or "barerrorbar".

add_line default FALSE; whether add line. Available when plot_type is "errorbar" or "barerrorbar".

line_size default 0.8; line size when add_line = TRUE. Available when plot_type is "errorbar" or "barerrorbar".

line_type default 2; an integer; line type when add_line = TRUE. The available case is same with line_size.

line_color default "grey50"; line color when add_line = TRUE. Available when by_group is NULL. Other available case is same with line_size.

line_alpha default 0.5; line transparency when add_line = TRUE. The available case is same with line_size.

heatmap_cell default "P.unadj"; the column of res_diff table for the cell of heatmap when formula with multiple factors is found in the method.

heatmap_sig default "Significance"; the column of res_diff for the significance label of heatmap.

heatmap_x default "Factors"; the column of res_diff for the x axis of heatmap.

heatmap_y default "Taxa"; the column of res_diff for the y axis of heatmap.

heatmap_lab_fill default "P value"; legend title of heatmap.

coefplot_sig_pos default 2; Significance label position in the coefficient point and errorbar plot. The formula is Estimate + coefplot_sig_pos * Std.Error. This plot is used when there is only one measure found in the table, and 'Estimate' and 'Std.Error' are both in the column names (such as for lm and lme methods). The x axis is 'Estimate', and y axis denotes 'Factors'. When coefplot_sig_pos is a negative value, the label is in the left of the errorbar. Errorbar size and width in the coefficient point plot can be adjusted with the parameters errorbar_size and errorbar_width. Point size and alpha can be adjusted with parameters point_size and point_alpha. The significance label size can be adjusted with parameter add_sig_text_size. Furthermore, the vertical line around 0 can be adjusted with parameters line_size, line_type, line_color and line_alpha.

... parameters passing to ggpubr::ggboxplot function (or other functions shown by plot_type parameter when it starts with "gg") or plot_cor function in trans_env class for the heatmap of multiple factors when formula is found in the res_diff of the object.

Returns: ggplot.

Examples:

```
\donttest{
t1 <- trans_alpha$new(dataset = dataset, group = "Group")
t1$cal_diff(method = "wilcox")
t1$plot_alpha(measure = "Shannon", add_sig = TRUE)
t1 <- trans_alpha$new(dataset = dataset, group = "Type", by_group = "Group")
t1$cal_diff(method = "wilcox")
t1$plot_alpha(measure = "Shannon", add_sig = 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")
t1 <- trans_alpha$new(dataset = dataset, group = "Type", by_group = "Group")
t1$cal_diff(method = "anova")


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


t1 <- trans_alpha$new(dataset = dataset, group = "Group")
t1$cal_diff(method = "wilcox")
t1$plot_alpha(measure = "Shannon", add_sig = TRUE)
t1 <- trans_alpha$new(dataset = dataset, group = "Type", by_group = "Group")
t1$cal_diff(method = "wilcox")
t1$plot_alpha(measure = "Shannon", add_sig = TRUE)
```

`trans_beta` *Create* `trans_beta` *object for beta-diversity analysis*

Description

This class is a wrapper for a series of beta-diversity related analysis, including ordination analysis 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>, ANOSIM and PERMDISP. Note that the beta diversity analysis methods related with environmental variables are encapsulated within the `trans_env` class.

Methods

Public methods:

- `trans_beta$new()`
- `trans_beta$cal_ordination()`
- `trans_beta$plot_ordination()`
- `trans_beta$cal_manova()`
- `trans_beta$cal_anosim()`
- `trans_beta$cal_betadisper()`
- `trans_beta$cal_group_distance()`
- `trans_beta$cal_group_distance_diff()`
- `trans_beta$plot_group_distance()`
- `trans_beta$plot_clustering()`
- `trans_beta$clone()`

Method `new()`:

Usage:

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

Arguments:

`dataset` an object of `microtable` class.

`measure` default NULL; a matrix name stored in `microtable$beta_diversity` list, such as "bray" or "jaccard", or a customized matrix; used for ordination, manova, group distance comparision, etc.; Please see `cal_betadiv` function of `microtable` class for more details.

`group` default NULL; a column name of `sample_table` in the input dataset; group information will be used for manova, betadisper or distance comparision.

Returns: `measure`, `group` and `dataset` stored in the object.

Examples:

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

Method `cal_ordination()`: Unconstrained ordination.

Usage:

```
trans_beta$cal_ordination(
  method = "PCoA",
  ncomp = 2,
  taxa_level = NULL,
  NMDS_matrix = TRUE,
```

```
  trans = FALSE,
  scale_species = FALSE,
  scale_species_ratio = 0.8,
  orthoI = NA,
  ordination = deprecated(),
  ...
)
```

Arguments:

`method` default "PCoA"; "PCoA", "NMDS", "PCA", "DCA", "PLS-DA" or "OPLS-DA". PCoA: principal coordinates analysis; NMDS: non-metric multidimensional scaling, PCA: principal component analysis; DCA: detrended correspondence analysis; PLS-DA: partial least squares discriminant analysis; OPLS-DA: orthogonal partial least squares discriminant analysis. For the methods details, please refer to the papers <doi:10.1111/j.1574-6941.2007.00375.x> (for PCoA, NMDS, PCA and DCA) and <doi:10.1186/s12859-019-3310-7> (for PLS-DA or OPLS-DA).

`ncomp` default 2; dimensions in the result. For the `method` option "PCA", "PCoA" or "DCA", the corresponding dimension information will be selected from the original model based on this parameter.. For all the dimension information, please refer to `model` in the results. For the `method` option "NMDS", this argument will be passed to the `k` parameter in the `vegan::metaMDS` function.

`taxa_level` default NULL; available for PCA, DCA or NMDS (`NMDS_matrix = TRUE`). Default NULL means using the `otu_table` in the microtable object. For other options, please provide the taxonomic rank names in `tax_table`, such as "Phylum" or "Genus". In such cases, the data will be merged according to the provided taxonomic levels to generated a new abundance table.

`NMDS_matrix` default TRUE; For the NMDS method, whether use a distance matrix as input like PCoA. If it is FALSE, the input will be the abundance table like PCA.

`trans` default FALSE; whether species abundance will be square root transformed; only available when `method` is "PCA" or "DCA". For method "NMDS" and `NMDS_matrix = FALSE`, please set the `autotransform` parameter, which will be passed to `vegan::metaMDS` function directly.

`scale_species` default FALSE; whether species loading in PCA, DCA or NMDS (`NMDS_matrix = FALSE`) is scaled.

`scale_species_ratio` default 0.8; the ratio to scale up the loading; multiply by the maximum distance between samples and origin. Only available when `scale_species = TURE`.

`orthoI` default NA; number of orthogonal components (for OPLS-DA only). Default NA means the number of orthogonal components is automatically computed. Please also see `orthoI` parameter in `opls` function of ropls package.

`ordination` deprecated. Please use `method` argument instead.

`...` parameters passed to `vegan::rda` function when `method = "PCA"`, or `vegan::decorana` function when `method = "DCA"`, or `ape::pcoa` function when `method = "PCoA"`, or `vegan::metaMDS` function when `method = "NMDS"`, or `ropls::opls` function when `method = "PLS-DA"` or `method = "OPLS-DA"` .

Returns: `res_ordination` list stored in the object. In the list, `model` is the original analysis results; `scores` is the sample scores table; `loading` is the feature loading table.

Examples:

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

Method `plot_ordination()`**:** Plot the ordination result.

Usage:

```
trans_beta$plot_ordination(
  plot_type = "point",
  choices = c(1, 2),
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  plot_color = NULL,
  plot_shape = NULL,
  plot_group_order = NULL,
  add_sample_label = NULL,
  point_size = 3,
  point_alpha = 0.8,
  centroid_segment_alpha = 0.6,
  centroid_segment_size = 1,
  centroid_segment_linetype = 3,
  ellipse_chull_fill = TRUE,
  ellipse_chull_alpha = 0.1,
  ellipse_level = 0.9,
  ellipse_type = "t",
  NMDS_stress_pos = c(1, 1),
  NMDS_stress_text_prefix = "",
  loading_arrow = FALSE,
  loading_taxa_num = 10,
  loading_text_taxlevel = NULL,
  loading_text_color = "black",
  loading_arrow_color = "grey30",
  loading_text_size = 3,
  loading_text_prefix = FALSE,
  loading_text_italic = FALSE
)
```

Arguments:

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

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

`choices` default c(1, 2); selected axis for the visualization; must be numeric vector. The maximum value must not exceed the parameter `ncomp` in the `cal_ordination` function.

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

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

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

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

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

add_sample_label default NULL; a column name in sample_table; If provided, show the point name in plot.

point_size default 3; point size when "point" is in plot_type parameter. point_size can also be a variable name in sample_table, such as "pH".

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

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

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

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

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

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

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

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

NMDS_stress_pos default c(1, 1); a numerical vector with two values used to represent the insertion position of the stress text. The first one denotes the x-axis, while the second one corresponds to the y-axis. The assigned position is determined by multiplying the respective value with the maximum point on the corresponding coordinate axis. Thus, the x-axis position is equal to max(points of x axis) * NMDS_stress_pos[1], and the y-axis position is equal to max(points of y axis) * NMDS_stress_pos[2]. Negative values can also be utilized for the negative part of the axis. NMDS_stress_pos = NULL denotes no stress text to show.

NMDS_stress_text_prefix default ""; If NMDS_stress_pos is not NULL, this parameter can be used to add text in front of the stress value.

loading_arrow default FALSE; whether show the loading using arrow.

loading_taxa_num default 10; the number of taxa used for the loading. Only available when loading_arrow = TRUE.

loading_text_taxlevel default NULL; which level of taxonomic table will be used. Default NULL means using the taxa_level parameter in the previous cal_ordination function.

loading_text_color default "black"; the color of taxa text. Only available when loading_arrow = TRUE.

loading_arrow_color default "grey30"; the color of taxa arrow. Only available when loading_arrow = TRUE.

loading_text_size default 3; the size of taxa text. Only available when loading_arrow = TRUE.

loading_text_prefix default FALSE; whether show the prefix (e.g., g__) in the taxa text. Only available when loading_arrow = TRUE.

loading_text_italic default FALSE; whether using italic for the taxa text. Only available when loading_arrow = TRUE.

Returns: ggplot.

Examples:

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

Method `cal_manova()`**:** Calculate perMANOVA (Permutational Multivariate Analysis of Variance) based on the adonis2 function of vegan package <doi:10.1111/j.1442-9993.2001.01070.pp.x>.

Usage:

```
trans_beta$cal_manova(
  manova_all = TRUE,
  manova_set = NULL,
  group = NULL,
  by_group = NULL,
  p_adjust_method = "fdr",
  by = "terms",
  by_auto_set = TRUE,
  permutations = 999,
  ...
)
```

Arguments:

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

`manova_set` default NULL; other specified group set for manova, such as `"Group + Type"` and `"Group*Type"`. Please also see the `formula` parameter (only right-hand side) in `adonis2` function of vegan package. The parameter manova_set has higher priority than manova_all parameter. If manova_set is provided; manova_all is disabled.

`group` default NULL; a column name of `sample_table` used for manova. If NULL, search `group` variable stored in the object. Available when `manova_set` is not provided.

`by_group` default NULL; one column name in `sample_table`; used to perform paired comparisions within each group. Only available when `manova_all = FALSE` and `manova_set` is not provided.

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

`by` default "terms"; same with the by parameter in `adonis2` function of vegan package.

`by_auto_set` default TRUE; Whether automatically set the options for by parameter ("marginal" or "terms") when `manova_set` is provided. The primary reason for setting this parameter is that using marginal effects (also known as "Type III" effects) is more robust for unbalanced experimental designs. Since the option `by = "margin"` in the `adonis2` function ignores main effects when interaction effects are present, we automatically set `by = "margin"` when there are no interaction effects, and set `by = "terms"` when interaction effects exist. If the user wants to use parameter by, please set `by_auto_set = FALSE`. Note that this parameter is only available when `manova_set` is provided.

`permutations` default 999; same with the `permutations` parameter in `adonis2` function of vegan package.

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

Returns: res_manova stored in object with data.frame class.

Examples:

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

Method `cal_anosim()`**:** Analysis of similarities (ANOSIM) based on the anosim function of vegan package.

Usage:

```
trans_beta$cal_anosim(
  paired = FALSE,
  group = NULL,
  by_group = NULL,
  p_adjust_method = "fdr",
  permutations = 999,
  ...
)
```

Arguments:

paired default FALSE; whether perform paired test between any two combined groups from all the input groups.

group default NULL; a column name of sample_table. If NULL, search group variable stored in the object.

by_group default NULL; one column name in sample_table; used to perform paired comparisions within each group. Only available when paired = TRUE.

p_adjust_method default "fdr"; p.adjust method; available when paired = TRUE; see method parameter of p.adjust function for available options.

permutations default 999; same with the permutations parameter in anosim function of vegan package.

... parameters passed to anosim function of vegan package.

Returns: res_anosim stored in object with data.frame class.

Examples:

```
t1$cal_anosim()
```

Method `cal_betadisper()`**:** Multivariate homogeneity test of groups dispersions (PERMDISP) based on betadisper function in vegan package.

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()`**:** Convert symmetric distance matrix to distance table of paired samples that are within groups or between groups.

Usage:

```
trans_beta$cal_group_distance(
  within_group = TRUE,
  by_group = NULL,
  ordered_group = NULL,
  sep = " vs "
)
```

Arguments:

`within_group` default TRUE; whether obtain distance table of paired samples within groups; if FALSE, obtain distances of paired samples between any two groups.

`by_group` default NULL; one colname name of `sample_table` in `microtable` object. If provided, transform distances by the provided by_group parameter. This is especially useful for ordering and filtering values further. When `within_group = TRUE`, the result of by_group parameter is the format of paired groups. When `within_group = FALSE`, the result of by_group parameter is the format same with the group information in `sample_table`.

`ordered_group` default NULL; a vector representing the ordered elements of `group` parameter; only useful when within_group = FALSE.

`sep` default TRUE; a character string to separate the group names after merging them into a new name.

Returns: `res_group_distance` stored in object.

Examples:

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

Method `cal_group_distance_diff()`**:** Differential test of converted distances across groups.

Usage:

```
trans_beta$cal_group_distance_diff(
  group = NULL,
  by_group = NULL,
  by_ID = NULL,
  ...
)
```

Arguments:

`group` default NULL; a column name of `object$res_group_distance` used for the statistics; If NULL, use the `group` inside the object.

`by_group` default NULL; a column of `object$res_group_distance` used to perform the differential test among elements in `group` parameter for each element in `by_group` parameter. So `by_group` has a larger scale than `group` parameter. This `by_group` is very different from the `by_group` parameter in the `cal_group_distance` function.

`by_ID` default NULL; a column of `object$res_group_distance` used to perform paired t test or paired wilcox test for the paired data, such as the data of plant compartments for different plant species (ID). So `by_ID` should be the smallest unit of sample collection without any repetition in it.

`...` parameters passed to `cal_diff` function of `trans_alpha` class.

Returns: `res_group_distance_diff` stored in object.

Examples:

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

Method `plot_group_distance()`**:** Plot the distances of paired groups within or between groups.

Usage:

```
trans_beta$plot_group_distance(plot_group_order = NULL, ...)
```

Arguments:

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

`...` parameters (except measure) passed to `plot_alpha` function of `trans_alpha` class.

Returns: `ggplot`.

Examples:

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

Method `plot_clustering()`**:** Plot clustering result based on the `ggdendro` package.

Usage:

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

Arguments:

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

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

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

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

Returns: `ggplot`.

Examples:

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

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

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

t1$cal_ordination(method = "PCoA")

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

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

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

t1$cal_manova(manova_all = TRUE)

## ------------------------------------------------
## Method `trans_beta$cal_anosim`
## ------------------------------------------------

t1$cal_anosim()

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

t1$cal_betadisper()

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


t1$cal_group_distance(within_group = TRUE)


## ------------------------------------------------
## Method `trans_beta$cal_group_distance_diff`
```

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


t1$cal_group_distance_diff()


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


t1$plot_group_distance()


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

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

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

Description

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

Author(s): Felipe Mansoldo and Chi Liu

Methods

Public methods:

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

- trans_classifier$cal_caretList()
- trans_classifier$cal_caretList_resamples()
- trans_classifier$plot_caretList_resamples()
- trans_classifier$clone()

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

Usage:

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

Arguments:

`dataset` an object of `microtable` class.

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

- **'Genus'** use Genus level table in `microtable$taxa_abund`, or other specific taxonomic rank, e.g., 'Phylum'. If an input level (e.g., ASV) is not found in the names of taxa_abund list, the function will use `otu_table` to calculate relative abundance of features.
- **'all'** use all the levels stored in `microtable$taxa_abund`.
- **other input** must be a data.frame object. It should have the same format with the tables in microtable$taxa_abund, i.e. rows are features; columns are samples with same names in sample_table.

`y.response` default NULL; the response variable in `sample_table` of input `microtable` object.

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

Returns: `data_feature` and `data_response` stored in the object.

Examples:

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

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

Usage:

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

Arguments:

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

Returns: `data_train` and `data_test` in the object.

Examples:

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

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

Usage:

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

Arguments:

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

Returns: `data_preProcess`, `data_train` and `data_test` in the object. `data_preProcess` is the return data generated by the `preProcess` function of caret package based on the training data. `data_train` and `data_test` are preprocessed training and testing data based on the `data_preProcess`.

Examples:

```
\dontrun{
# "nzv" removes near zero variance predictors
t1$cal_preProcess(method = c("center", "scale", "nzv"))
}
```

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

Usage:

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

Arguments:

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

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

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

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

Returns: optimized `data_train` and `data_test` in the object.

Examples:

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

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

Usage:

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

Arguments:

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

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

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

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

Returns: `trainControl` in the object.

Examples:

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

Method `cal_train()`: Run the model training. Please see https://topepo.github.io/caret/available-models.html for available models.

Usage:

```
trans_classifier$cal_train(method = "rf", max.mtry = 2, ntree = 500, ...)
```

Arguments:

`method` default "rf"; "rf": random forest; see method in `train` function of caret package for other options. For method = "rf", the `tuneGrid` is set: `expand.grid(mtry = seq(from = 1, to = max.mtry))`

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

`ntree` default 500; for method = "rf"; Number of trees to grow. The default 500 is same with the `ntree` parameter in `randomForest` function in randomForest package. When it is a vector with more than one element, the function will try to optimize the model to select a best one, such as `c(100, 500, 1000)`.

`...` parameters pass to `caret::train` function.

Returns: `res_train` in the object.

Examples:

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

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

Usage:

```
trans_classifier$cal_feature_imp(rf_feature_sig = FALSE, ...)
```

Arguments:

`rf_feature_sig` default FALSE; whether calculate feature significance in 'rf' model using `rfPermute` package; only available for `method = "rf"` in `cal_train` function.

`...` parameters pass to `varImp` function of caret package. If `rf_feature_sig` is TURE and `train_method` is "rf", the parameters will be passed to `rfPermute` function of rfPermute package.

Returns: `res_feature_imp` in the object. One row for each predictor variable. The column(s) are different importance measures. For the method 'rf', it is MeanDecreaseGini (classification) or IncNodePurity (regression) when `rf_feature_sig = FALSE`.

Examples:

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

Method `plot_feature_imp()`**:** Bar plot for feature importance.

Usage:

```
trans_classifier$plot_feature_imp(
  rf_sig_show = NULL,
  show_sig_group = FALSE,
  ...
)
```

Arguments:

`rf_sig_show` default NULL; "MeanDecreaseAccuracy" (Default) or "MeanDecreaseGini" for random forest classification; "%IncMSE" (Default) or "IncNodePurity" for random forest regression; Only available when `rf_feature_sig = TRUE` in function `cal_feature_imp`, which generate "MeanDecreaseGini" (and "MeanDecreaseAccuracy") or "%IncMSE" (and "IncNodePurity") in the column names of `res_feature_imp`; Function can also generate "Significance" according to the p value.

`show_sig_group` default FALSE; whether show the features with different significant groups; Only available when "Significance" is found in the data.

`...` parameters pass to `plot_diff_bar` function of `trans_diff` package.

Returns: `ggplot2` object.

Examples:

```
\dontrun{
t1$plot_feature_imp(use_number = 1:20, coord_flip = FALSE)
}
```

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

Usage:

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

Arguments:

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

Returns: `res_predict`, `res_confusion_fit` and `res_confusion_stats` stored in the object. The `res_predict` is the predicted result for `data_test`. Several evaluation metrics in `res_confusion_fit` are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Sensitivity = Recall = TPR = \frac{TP}{TP + FN}$$

$$Specificity = TNR = 1 - FPR = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Prevalence = \frac{TP + FN}{TP + TN + FP + FN}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$Kappa = \frac{Accuracy - Pe}{1 - Pe}$$

where TP is true positive; TN is ture negative; FP is false positive; and FN is false negative; FPR is False Positive Rate; TPR is True Positive Rate; TNR is True Negative Rate; Pe is the hypothetical probability of chance agreement on the classes for reference and prediction in the confusion matrix. Accuracy represents the ratio of correct predictions. Precision identifies how the model accurately predicted the positive classes. Recall (sensitivity) measures the ratio of actual positives that are correctly identified by the model. F1-score is the weighted average score of recall and precision. The value at 1 is the best performance and at 0 is the worst. Prevalence represents how often positive events occurred. Kappa identifies how well the model is predicting.

Examples:

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

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

Usage:

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

Arguments:

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

`plot_statistics` default TRUE; whether plot the statistics.

Returns: `ggplot` object.

Examples:

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

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

Usage:

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

Arguments:

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

Returns: a list `res_ROC` stored in the object. It has two tables: `res_roc` and `res_pr`. AUC: Area Under the ROC Curve. For the definition of metrics, please refer to the return part of function `cal_predict`.

Examples:

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

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

Usage:

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

Arguments:

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

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

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

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

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

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

Returns: `ggplot2` object.

Examples:

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

Method `cal_caretList()`**:** Use `caretList` function of caretEnsemble package to run multiple models. For the available models, please run `names(getModelInfo())`.

Usage:

```
trans_classifier$cal_caretList(...)
```

Arguments:

`...` parameters pass to `caretList` function of `caretEnsemble` package.

Returns: `res_caretList_models` in the object.

Examples:

```
\dontrun{
t1$cal_caretList(methodList = c('rf', 'svmRadial'))
}
```

Method `cal_caretList_resamples()`**:** Use `resamples` function of caret package to collect the metric values based on the `res_caretList_models` data.

Usage:

```
trans_classifier$cal_caretList_resamples(...)
```

Arguments:

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

Returns: `res_caretList_resamples` list and `res_caretList_resamples_reshaped` table in the object.

Examples:

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

Method `plot_caretList_resamples()`**:** Visualize the metric values based on the `res_caretList_resamples_reshape` data.

Usage:

```
trans_classifier$plot_caretList_resamples(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  ...
)
```

Arguments:

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

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

Returns: ggplot object.

Examples:

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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


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


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

## Not run:
t1$cal_split(prop.train = 3/4)

## End(Not run)

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

## Not run:
# "nzv" removes near zero variance predictors
t1$cal_preProcess(method = c("center", "scale", "nzv"))

## End(Not run)

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

## Not run:
```

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

## End(Not run)

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

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

## End(Not run)

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

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

## End(Not run)

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

## Not run:
t1$cal_feature_imp()

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$plot_feature_imp`
## ------------------------------------------------

## Not run:
t1$plot_feature_imp(use_number = 1:20, coord_flip = FALSE)

## End(Not run)

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

## Not run:
t1$cal_predict()

## End(Not run)

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

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

## Not run:
t1$plot_confusionMatrix()

## End(Not run)

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

## Not run:
t1$cal_ROC()

## End(Not run)

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

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

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_caretList`
## ------------------------------------------------

## Not run:
t1$cal_caretList(methodList = c('rf', 'svmRadial'))

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$cal_caretList_resamples`
## ------------------------------------------------

## Not run:
t1$cal_caretList_resamples()

## End(Not run)

## ------------------------------------------------
## Method `trans_classifier$plot_caretList_resamples`
## ------------------------------------------------

## Not run:
t1$plot_caretList_resamples()

## End(Not run)
```

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

Description

This class is a wrapper for a series of differential abundance test and indicator analysis methods, including LEfSe based on the Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>, random forest <doi:10.1016/j.geoderma.2018.09.035>, metastat based on White et al. (2009) <doi:10.1371/journal.pcbi.1000352>, non-parametric Kruskal-Wallis Rank Sum Test, Dunn's Kruskal-Wallis Multiple Comparisons based on the FSA package, Wilcoxon Rank Sum and Signed Rank Tests, t-test, anova, Scheirer Ray Hare test, R package metagenomeSeq Paulson et al. (2013) <doi:10.1038/nmeth.2658>, R package ANCOMBC <doi:10.1038/s41467-020-17041-7>, R package ALDEx2 <doi:10.1371/journal.pone.0067019; 10.1186/2049-2618-2-15>, R package MicrobiomeStat <doi:10.1186/s13059-022-02655-5>, beta regression <doi:10.18637/jss.v034.i02>, R package maaslin2, linear mixed-effects model and generalized linear mixed model.

Methods

Public methods:

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

Method new():

Usage:

```
trans_diff$new(
  dataset = NULL,
 method = c("lefse", "rf", "metastat", "metagenomeSeq", "KW", "KW_dunn", "wilcox",
  "t.test", "anova", "scheirerRayHare", "lm", "ancombc2", "ALDEx2_t", "ALDEx2_kw",
  "DESeq2", "edgeR", "linda", "maaslin2", "betareg", "lme", "glmm", "glmm_beta")[1],
  group = NULL,
  taxa_level = "all",
  filter_thres = 0,
  alpha = 0.05,
  p_adjust_method = "fdr",
  transformation = NULL,
  remove_unknown = TRUE,
  lefse_subgroup = NULL,
  lefse_min_subsam = 10,
  lefse_sub_strict = FALSE,
  lefse_sub_alpha = NULL,
  lefse_norm = 1e+06,
  nresam = 0.6667,
```

```
  boots = 30,
  rf_imp_type = 2,
  group_choose_paired = NULL,
  metagenomeSeq_count = 1,
  ALDEx2_sig = c("wi.eBH", "kw.eBH"),
  by_group = NULL,
  by_ID = NULL,
  beta_pseudo = .Machine$double.eps,
  ...
)
```

Arguments:

`dataset` default NULL; `microtable` object.

`method` default "lefse". Some methods (e.g., "lefse", "KW", "wilcox", "anova", "lm", "betareg", "glmm" and "glmm_beta") invoke the `taxa_abund` list (generally relative abundance data) of input microtable object for the analysis. Some (e.g., "metastat", "metagenomeSeq", "ALDEx2_t", "DESeq2", "edgeR", "ancombc2" and "linda") use the `otu_table` of input microtable object for the analysis. Available options include:

'lefse' LEfSe method based on Segata et al. (2011) <doi:10.1186/gb-2011-12-6-r60>

'rf' random forest and non-parametric test method based on An et al. (2019) <doi:10.1016/j.geoderma.2018.09.035>

'metastat' Metastat method for all paired groups based on White et al. (2009) <doi:10.1371/journal.pcbi.1000352>

'metagenomeSeq' zero-inflated log-normal model-based differential test method from `metagenomeSeq` package.

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

'KW_dunn' Dunn's Kruskal-Wallis Multiple Comparisons when group number > 2; see dunnTest function in `FSA` package

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

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

'anova' ANOVA for one-way or multi-factor analysis; see `cal_diff` function of `trans_alpha` class

'scheirerRayHare' Scheirer Ray Hare test for nonparametric test used for a two-way factorial experiment; see `scheirerRayHare` function of `rcompanion` package

'lm' Linear Model based on the `lm` function

'ALDEx2_t' runs Welch's t and Wilcoxon tests with `ALDEx2` package; see also the test parameter in `ALDEx2::aldex` function; ALDEx2 uses the centred log-ratio (clr) transformation and estimates per-feature technical variation within each sample using Monte-Carlo instances drawn from the Dirichlet distribution; Reference: <doi:10.1371/journal.pone.0067019> and <doi:10.1186/2049-2618-2-15>; require `ALDEx2` package to be installed (https://bioconductor.org/packages/rel

'ALDEx2_kw' runs Kruskal-Wallace and generalized linear model (glm) test with `ALDEx2` package; see also the `test` parameter in `ALDEx2::aldex` function.

'DESeq2' Differential expression analysis based on the Negative Binomial (a.k.a. Gamma-Poisson) distribution based on the `DESeq2` package.

'edgeR' The `exactTest` method of `edgeR` package is implemented.

'ancombc2' Analysis of Compositions of Microbiomes with Bias Correction (ANCOM-BC) based on the `ancombc2` function from `ANCOMBC` package. If the `fix_formula` parameter is not provided, the function can automatically assign it by using group parameter. For this method, the `group` parameter is directly passed to the group parameter of

ancombc2 function. Reference: <doi:10.1038/s41467-020-17041-7><10.1038/s41592-023-02092-7>; Require ANCOMBC package to be installed (https://bioconductor.org/packages/release/bioc/html/ANC

'linda' Linear Model for Differential Abundance Analysis of High-dimensional Compositional Data based on the linda function of MicrobiomeStat package. For linda method, please provide either the group parameter or the formula parameter. When the formula parameter is provided, it should start with '~' as it is directly used by the linda function. If the group parameter is used, the prefix '~' is not necessary as the function can automatically add it. The parameter feature.dat.type = 'count' has been fixed. Other parameters can be passed to the linda function. Reference: <doi:10.1186/s13059-022-02655-5>

'maaslin2' finding associations between metadata and potentially high-dimensional microbial multi-omics data based on the Maaslin2 package. Using this option can invoke the trans_env$cal_cor function with method = "maaslin2".

'betareg' Beta Regression based on the betareg package. Please see the beta_pseudo parameter for the use of pseudo value when there is 0 or 1 in the data

'lme' Linear Mixed Effect Model based on the lmerTest package. In the return table, the significance of fixed factors are tested by function anova. The significance of 'Estimate' in each term of fixed factors comes from the model.

'glmm' Generalized linear mixed model (GLMM) based on the glmmTMB package. The formula and family parameters are needed. Please refer to glmmTMB package to select the family function, e.g. family = glmmTMB::lognormal(link = "log"). The usage of formula is similar with that in 'lme' method. For more available parameters, please see glmmTMB::glmmTMB function and use parameter passing. In the result, Conditional R2 and Marginal R2 represent the variance explained by both fixed and random effects and the variance explained by fixed effects, respectively. For more details on R2 calculation, please refer to the article <doi: 10.1098/rsif.2017.0213>. The significance of fixed factors are tested by Chi-square test from function car::Anova. The significance of 'Estimate' in each term of fixed factors comes from the model.

'glmm_beta' Generalized linear mixed model with a family function of beta distribution, developed for the relative abundance (ranging from 0 to 1) of taxa specifically. This is an extension of the GLMM model in 'glmm' option. The only difference is in glmm_beta the family function is fixed with the beta distribution function, i.e. family = glmmTMB::beta_family(link = "logit"). Please see the beta_pseudo parameter for the use of pseudo value when there is 0 or 1 in the data

group default NULL; sample group used for the comparision; a colname of input microtable$sample_table; It is necessary when method is not "anova" or method is "anova" but formula is not provided. Once group is provided, the return res_abund will have mean and sd values for group.

taxa_level default "all"; 'all' represents using abundance data of all taxonomic ranks; For testing at a specific rank, provide taxonomic rank name, such as "Genus". If the provided taxonomic name is neither 'all' nor a colname in tax_table of input dataset (e.g., "ASV"), the function will use the features in input microtable$otu_table automatically. Note that a specific level (e.g., "ASV") should be provided for method: "metastat", "metagenomeSeq", "ALDEx2_t", "DESeq2", "edgeR", "ancombc2", "linda", "maaslin2".

filter_thres default 0; the abundance threshold, such as 0.0005 when the input is relative abundance; only available when method != "metastat". The features with abundances lower than filter_thres will be filtered.

alpha default 0.05; significance threshold to select taxa when method is "lefse" or "rf"; or

used to generate significance letters when method is 'anova' or 'KW_dunn' like the alpha parameter in `cal_diff` of `trans_alpha` class.

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of `p.adjust` function for other available options; "none" means disable p value adjustment; So when `p_adjust_method = "none"`, P.adj is same with P.unadj.

`transformation` default NULL; feature abundance transformation method in the class `trans_norm`, such as 'AST' for the arc sine square root transformation. Only available when `method` is one of "KW", "KW_dunn", "wilcox", "t.test", "anova", "scheirerRayHare", "betareg" and "lme".

`remove_unknown` default TRUE; whether remove unknown features that donot have clear classification information.

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

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

`lefse_sub_strict` default FALSE; whether remove the features strictly in the sub-checking. FALSE means only removing the features that have different orders of medians across subgroups with those across groups and the statistics are also significant. TRUE means removing the features that are not significant in one (or more) sub-test or have different orders of medians across sub-groups with those across groups.

`lefse_sub_alpha` default NULL; The significance threshold in the test for lefse sub-groups. NULL means it is same with `alpha`.

`lefse_norm` default 1000000; normalization value used in lefse to scale abundances for each level. A `lefse_norm` value < 0 (e.g., -1) means no normalization same with the LEfSe python version.

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

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

`rf_imp_type` default 2; the type of feature importance in random forest when `method = "rf"`. Same with `type` parameter in `importance` function of `randomForest` package. 1=mean decrease in accuracy (MeanDecreaseAccuracy), 2=mean decrease in node impurity (MeanDecreaseGini).

`group_choose_paired` default NULL; a vector used for selecting the required groups for paired testing instead of all paired combinations across groups; Available when method is "metastat", "metagenomeSeq", "ALDEx2_t" or "edgeR".

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

`ALDEx2_sig` default c("wi.eBH", "kw.eBH"); which column of the final result is used as the significance asterisk assignment; applied to method = "ALDEx2_t" or "ALDEx2_kw"; the first element is provided to "ALDEx2_t"; the second is provided to "ALDEx2_kw"; for "ALDEx2_t", the available choice is "wi.eBH" (Expected Benjamini-Hochberg corrected P value of Wilcoxon test) and "we.eBH" (Expected BH corrected P value of Welch's t test); for "ALDEx2_kw"; for "ALDEx2_t", the available choice is "kw.eBH" (Expected BH corrected P value of Kruskal-Wallace test) and "glm.eBH" (Expected BH corrected P value of glm test).

`by_group` default NULL; a column of sample_table used to perform the differential test among groups (`group` parameter) for each group (`by_group` parameter). So `by_group` has a higher

level than group parameter. Same with the by_group parameter in trans_alpha class. Only available when method is one of c("KW", "KW_dunn", "wilcox", "t.test", "anova", "scheirerRayHare").

by_ID default NULL; a column of sample_table used to perform paired t test or paired wilcox test for the paired data, such as the data of plant compartments for different plant species (ID). So by_ID in sample_table should be the smallest unit of sample collection without any repetition in it. Same with the by_ID parameter in trans_alpha class.

beta_pseudo default .Machine$double.eps; the pseudo value used when the parameter method is 'betareg' or 'glmm_beta'. As the beta distribution function limits 0 < response value < 1, a pseudo value will be added for the data that equal to 0. The data that equal to 1 will be replaced by 1/(1 + beta_pseudo).

... parameters passed to cal_diff function of trans_alpha class when method is one of "KW", "KW_dunn", "wilcox", "t.test", "anova", "betareg", "lme", "glmm" or "glmm_beta"; passed to randomForest::randomForest function when method = "rf"; passed to ANCOMBC::ancombc2 function when method is "ancombc2" (except tax_level, global and fix_formula parameters); passed to ALDEx2::aldex function when method = "ALDEx2_t" or "ALDEx2_kw"; passed to DESeq2::DESeq function when method = "DESeq2"; passed to MicrobiomeStat::linda function when method = "linda"; passed to trans_env$cal_cor function when method = "maaslin2".

Returns: res_diff and res_abund.
res_abund includes mean abundance of each taxa (Mean), standard deviation (SD), standard error (SE) and sample number (N) in the group (Group).
res_diff is the detailed differential test result depending on the method choice, may containing:
"Comparison": The groups for the comparision, maybe all groups or paired groups. If this column is not found, all groups are used;
"Group": Which group has the maximum median or mean value across the test groups; For non-parametric methods, median value; For t.test, mean value;
"Taxa": which taxa is used in this comparision;
"Method": Test method used in the analysis depending on the method input;
"LDA" or others: LDA: linear discriminant score in LEfSe; MeanDecreaseAccuracy and MeanDecreaseGini: mean decreasing in accuracy or in node impurity (gini index) in random forest;
"P.unadj": original p value;
"P.adj": adjusted p value;
"Estimate" and "Std.Error": When method is "betareg", "lm", "lme" or "glmm", "Estimate" and "Std.Error" represent fitted coefficient and its standard error, respectively;
Others: qvalue: qvalue in metastat analysis.

Examples:

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

Method plot_diff_abund()**:** Plot the abundance of taxa.

The significance can be optionally added in the plot. The taxa displayed are based on the taxa in

the 'res_diff' table, selected using parameters. If the user filters out the non-significant taxa from the 'res_diff' table, these taxa will also be filtered from the plot.

Usage:

```
trans_diff$plot_diff_abund(
  use_number = 1:10,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  select_taxa = NULL,
  simplify_names = TRUE,
  keep_prefix = TRUE,
  group_order = NULL,
  order_x_mean = FALSE,
  coord_flip = TRUE,
  add_sig = TRUE,
  xtext_angle = 45,
  xtext_size = 13,
  ytitle_size = 17,
  ...
)
```

Arguments:

`use_number` default 1:10; numeric vector; the sequences of taxa (1:n) selected in the plot; If n is larger than the number of total significant taxa, automatically use the total number as n.

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

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

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

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

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

`order_x_mean` default FALSE; whether order the taxa in x axis by the means of abundances from large to small. If `TRUE`, all other factors in the data will become invalid.

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

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

`xtext_angle` default 45; number (e.g. 45). Angle of text in x axis.

`xtext_size` default 13; x axis text size. NULL means the default size in ggplot2. If `coord_flip = TRUE`, it represents the text size of the y axis.

`ytitle_size` default 17; y axis title size. If `coord_flip = TRUE`, it represents the title size of the x axis (i.e. "Relative abundance").

`...` parameters passed to `plot_alpha` function of `trans_alpha` class.

Returns: ggplot.

Examples:

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

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

Method `plot_diff_bar()`**:** Bar plot for indicator index, such as LDA score and P value.

Usage:

```
trans_diff$plot_diff_bar(
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  color_group_map = FALSE,
  use_number = 1:10,
  threshold = NULL,
  select_group = NULL,
  keep_full_name = FALSE,
  keep_prefix = TRUE,
  group_order = NULL,
  group_aggre = TRUE,
  group_two_sep = TRUE,
  coord_flip = TRUE,
  add_sig = FALSE,
  add_sig_increase = 0.1,
  add_sig_text_size = 5,
  xtext_angle = 45,
  xtext_size = 10,
  ytext_size = NULL,
  axis_text_y = deprecated(),
  heatmap_cell = "P.unadj",
  heatmap_sig = "Significance",
  heatmap_x = "Factors",
  heatmap_y = "Taxa",
  heatmap_lab_fill = "P value",
  ...
)
```

Arguments:

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

`color_group_map` default FALSE; whether match the colors to groups in order to fix the color in each group when part of groups are not shown in the plot. When `color_group_map = TRUE`, the group_order inside the object will be used as full groups set to guide the color extraction.

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

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

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

`keep_full_name` default FALSE; whether keep the taxonomic full lineage names.

`keep_prefix` default TRUE; whether retain the taxonomic prefix, such as "g__".

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

`group_aggre` default TRUE; whether aggregate the features for each group.

`group_two_sep` default TRUE; whether display the features of two groups on opposite sides of the coordinate axes when there are only two groups in total.

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

`add_sig` default FALSE; whether add significance label (asterisk) above the bar.

`add_sig_increase` default 0.1; the axis position (`Value + add_sig_increase * max(Value)`) from which to add the significance label; only available when `add_sig = TRUE`.

`add_sig_text_size` default 5; the size of added significance label; only available when `add_sig = TRUE`.

`xtext_angle` default 45; number ranging from 0 to 90; used to make x axis text generate angle to reduce text overlap; only available when coord_flip = FALSE.

`xtext_size` default 10; text size of x axis.

`ytext_size` default NULL; text size of y axis. NULL means default ggplot2 value.

`axis_text_y` deprecated. Please use `ytext_size` argument instead.

`heatmap_cell` default "P.unadj"; the column of data for the cell of heatmap when formula with multiple factors is found in the method.

`heatmap_sig` default "Significance"; the column of data for the significance label of heatmap.

`heatmap_x` default "Factors"; the column of data for the x axis of heatmap.

`heatmap_y` default "Taxa"; the column of data for the y axis of heatmap.

`heatmap_lab_fill` default "P value"; legend title of heatmap.

`...` parameters passing to `geom_bar` for the bar plot or `plot_cor` function in `trans_env` class for the heatmap of multiple factors when formula is found in the method.

Returns: ggplot.

Examples:

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

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

Usage:

```
trans_diff$plot_diff_cladogram(
  color = RColorBrewer::brewer.pal(8, "Dark2"),
  group_order = NULL,
  use_taxa_num = 200,
  filter_taxa = NULL,
```

```
  use_feature_num = NULL,
  clade_label_level = 4,
  select_show_labels = NULL,
  only_select_show = FALSE,
  sep = "|",
  branch_size = 0.2,
  alpha = 0.2,
  clade_label_size = 2,
  clade_label_size_add = 5,
  clade_label_size_log = exp(1),
  node_size_scale = 1,
  node_size_offset = 1,
  annotation_shape = 22,
  annotation_shape_size = 5
)
```

Arguments:

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

`group_order` default NULL; a vector to order the legend in plot; If NULL, the function can first check whether the group column of sample_table is factor. If yes, use the levels in it. If provided, this parameter can overwrite the levels in the group of sample_table. If the number of provided group_order is less than the number of groups in `res_diff$Group`, the function will select the groups of group_order automatically.

`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 metric (method = "lefse" or "rf") from high to low.

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

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

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

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

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

`alpha` default 0.2; shading of the color.

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

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

`clade_label_size_log` default `exp(1)`; the base of `log` function for added size of the clade label; the size formula: `clade_label_size + log(clade_label_level + clade_label_size_add, base = clade_label_size_log)`; so use `clade_label_size_log`, `clade_label_size_add` and `clade_label_size` can totally control the label size for different taxonomic levels.

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

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

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

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

Returns: ggplot.

Examples:

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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


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


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


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


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

```
t1$plot_diff_bar(use_number = 1:20)


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

## Not run:
t1$plot_diff_cladogram(use_taxa_num = 100, use_feature_num = 30, select_show_labels = NULL)

## End(Not run)
```

`trans_env` *Create* `trans_env` *object to analyze the association between environmental factor and microbial community.*

Description

This class is a wrapper for a series of operations associated with environmental measurements, including redundancy analysis, mantel test, correlation analysis and linear fitting.

Methods

Public methods:

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

Method `new()`:

Usage:

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

Arguments:

`dataset` the object of `microtable` Class.

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

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

`character2numeric` default FALSE; whether convert all the character or factor columns to numeric type using the `dropallfactors` function. If TRUE, character columns will first be attempted to convert to numeric. If that fails, they will be converted to the factor type and then to numeric.

`standardize` default FALSE; whether scale environmental variables to zero mean and unit variance.

`complete_na` default FALSE; Whether fill the NA (missing value) in the environmental data; If TRUE, the function can run the interpolation with the `mice` package.

Returns: `data_env` stored in the object.

Examples:

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

Method `cal_diff()`**:** Differential test of environmental variables across groups.

Usage:

```
trans_env$cal_diff(
  group = NULL,
  by_group = NULL,
 method = c("KW", "KW_dunn", "wilcox", "t.test", "anova", "scheirerRayHare", "lm",
    "lme", "glmm")[1],
  ...
)
```

Arguments:

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

`by_group` default NULL; perform differential test among groups (`group` parameter) within each group (`by_group` parameter).

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

- **'KW'** KW: Kruskal-Wallis Rank Sum Test for all groups (>= 2)
- **'KW_dunn'** Dunn's Kruskal-Wallis Multiple Comparisons, see `dunnTest` function in `FSA` package
- **'wilcox'** Wilcoxon Rank Sum and Signed Rank Tests for all paired groups
- **'t.test'** Student's t-Test for all paired groups
- **'anova'** Duncan's new multiple range test for one-way anova; see `duncan.test` function of `agricolae` package. For multi-factor anova, see `aov`
- **'scheirerRayHare'** Scheirer Ray Hare test for nonparametric test used for a two-way factorial experiment; see `scheirerRayHare` function of `rcompanion` package
- **'lm'** Linear model based on the `lm` function
- **'lme'** lme: Linear Mixed Effect Model based on the `lmerTest` package. The `formula` parameter should be provided.
- **'glmm'** Generalized linear mixed model (GLMM) based on the glmmTMB package. The `formula` and `family` parameters are needed. Please refer to glmmTMB package to select the family function, e.g. `family = glmmTMB::lognormal(link = "log")`. The usage of formula is similar with that in 'lme' method. For the details of return table, please refer to the help document of trans_diff class.

`...` parameters passed to `cal_diff` function of `trans_alpha` class.

Returns: `res_diff` stored in the object. In the data frame, 'Group' column means that the group has the maximum median or mean value across the test groups; For non-parametric methods, median value; For t.test, mean value.

Examples:

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

Method `plot_diff()`**:** Plot environmental variables across groups and add the significance label.

Usage:

```
trans_env$plot_diff(...)
```

Arguments:

`...` parameters passed to `plot_alpha` in `trans_alpha` class. Please see `plot_alpha` function of `trans_alpha` for all the available parameters.

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

Usage:

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

Arguments:

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

`ggpairs` default TRUE; whether use `GGally::ggpairs` function to plot the correlation results. If `ggpairs = FALSE`, the function will output a table with all the values instead of a graph. In this case, the function will call `cal_cor` to calculate autocorrelation instead of using the ggpairs function in GGally, so please use parameter passing to control more options.

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

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

`...` parameters passed to `GGally::ggpairs` when `ggpairs = TRUE` or passed to `cal_cor` of `trans_env` class when `ggpairs = FALSE`.

Returns: `ggmatrix` when `ggpairs = TRUE` or data.frame object when `ggpairs = FALSE`.

Examples:

```
\dontrun{
# Spearman correlation
t1$cal_autocor(upper = list(continuous = GGally::wrap("cor", method= "spearman")))
}
```

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

Usage:

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

Arguments:

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

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

`taxa_level` default NULL; the taxonomic level used in RDA or CCA. Default NULL means using the merged data at "Genus" level. "ASV" or "OTU" can also be provided for the use of `otu_table` in microtable object.

`taxa_filter_thres` default NULL; relative abundance threshold used to filter taxa when method is "RDA" or "CCA".

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

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

`...` paremeters passed to `dbrda`, `rda` or `cca` function according to the `method` parameter.

Returns: `res_ordination` and `res_ordination_R2` stored in the object.

Examples:

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

Method `cal_ordination_anova()`**:** Use anova to test the significance of the terms and axis in ordination.

Usage:

```
trans_env$cal_ordination_anova(...)
```

Arguments:

`...` parameters passed to `anova` function.

Returns: `res_ordination_terms` and `res_ordination_axis` stored in the object.

Examples:

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

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

Usage:

```
trans_env$cal_ordination_envfit(...)
```

Arguments:

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

Returns: `res_ordination_envfit` stored in the object.

Examples:

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

Method `trans_ordination()`**:** Transform ordination results for the following plot.

Usage:

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

Arguments:

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

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

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

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

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

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

Returns: `res_ordination_trans` stored in the object.

Examples:

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

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

Usage:

```
trans_env$plot_ordination(
  plot_color = NULL,
  plot_shape = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  env_text_color = "black",
  env_arrow_color = "grey30",
  taxa_text_color = "firebrick1",
  taxa_arrow_color = "firebrick1",
  env_text_size = 3.7,
  taxa_text_size = 3,
  taxa_text_prefix = FALSE,
  taxa_text_italic = TRUE,
  plot_type = "point",
  point_size = 3,
  point_alpha = 0.8,
  centroid_segment_alpha = 0.6,
  centroid_segment_size = 1,
  centroid_segment_linetype = 3,
  ellipse_chull_fill = TRUE,
  ellipse_chull_alpha = 0.1,
  ellipse_level = 0.9,
  ellipse_type = "t",
  add_sample_label = NULL,
  env_nudge_x = NULL,
  env_nudge_y = NULL,
  taxa_nudge_x = NULL,
```

```
  taxa_nudge_y = NULL,
  ...
)
```

Arguments:

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

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

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

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

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

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

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

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

`env_text_size` default 3.7; environmental variable text size.

`taxa_text_size` default 3; taxa text size.

`taxa_text_prefix` default FALSE; whether show the prefix (e.g., g__) of taxonomic information in the text.

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

`plot_type` default "point"; plotting type of samples; one or more elements of "point", "ellipse", "chull", "centroid" and "none"; "none" denotes nothing.

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

`point_size` default 3; point size in plot when "point" is in `plot_type`. `point_size` can also be a variable name in `sample_table`, such as "pH".

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

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

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

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

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

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

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

`ellipse_type` default "t"; ellipse type when "ellipse" is in `plot_type`; see type parameter in `stat_ellipse` function of ggplot2 package.

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

`env_nudge_x` default NULL; numeric vector to adjust the env text x axis position; passed to nudge_x parameter of `ggrepel::geom_text_repel` function; default NULL represents automatic adjustment; the length must be same with the row number of `object$res_ordination_trans$df_arrows`.

For example, if there are 5 env variables, env_nudge_x should be something like `c(0.1, 0, -0.2, 0, 0)`. Note that this parameter and env_nudge_y is generally used when the automatic text adjustment is not very well.

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

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

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

`...` paremeters passed to `geom_point` for controlling sample points.

Returns: ggplot object.

Examples:

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

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

Usage:

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

Arguments:

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

add_matrix default NULL; additional distance matrix provided when the beta diversity matrix in the dataset is not used.

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

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

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

by_group default NULL; one column name or number in sample_table; used to perform mantel test for different groups separately.

... paremeters passed to mantel of vegan package.

Returns: res_mantel in object.

Examples:

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

Method `cal_cor()`**:** Calculate the correlations between taxonomic abundance and environmental variables. Actually, it can also be applied to other correlation between any two variables from two tables.

Usage:

```
trans_env$cal_cor(
  use_data = c("Genus", "all", "other")[1],
  method = c("pearson", "spearman", "kendall", "maaslin2")[1],
  partial = FALSE,
  partial_fix = NULL,
  add_abund_table = NULL,
  filter_thres = 0,
  use_taxa_num = NULL,
  other_taxa = NULL,
  p_adjust_method = "fdr",
  p_adjust_type = c("All", "Taxa", "Env")[1],
  by_group = NULL,
  group_use = NULL,
  group_select = NULL,
  taxa_name_full = TRUE,
  tmp_input_maaslin2 = "tmp_input",
  tmp_output_maaslin2 = "tmp_output",
  cor_method = deprecated(),
  ...
)
```

Arguments:

use_data default "Genus"; "Genus", "all" or "other"; "Genus" or other taxonomic names (e.g., "Phylum", "ASV"): invoke taxonomic abundance table in taxa_abund list of the microtable object; "all": merge all the taxonomic abundance tables in taxa_abund list into one; "other": provide additional taxa names by assigning other_taxa parameter.

`method` default "pearson"; "pearson", "spearman", "kendall" or "maaslin2"; correlation method. "pearson", "spearman" or "kendall" all refer to the correlation analysis based on the `cor.test` function in R. "maaslin2" is the method in `Maaslin2` package for finding associations between metadata and potentially high-dimensional microbial multi-omics data.

`partial` default FALSE; whether perform partial correlation based on the `ppcor` package. Available when `method` is "pearson", "spearman" or "kendall".

`partial_fix` default NULL; selected environmental variable names used as third group of variables in all the partial correlations. If NULL; all the variables (except the one for correlation) in the environmental data will be used as the third group of variables. Otherwise, the function will control for the provided variables (one or more) in all the partial correlations, and the variables in `partial_fix` will not be employed anymore in the correlation analysis.

`add_abund_table` default NULL; additional data table to be used. Row names must be sample names.

`filter_thres` default 0; the abundance threshold, such as 0.0005 when the input is relative abundance. The features with abundances lower than filter_thres will be filtered. This parameter cannot be applied when add_abund_table parameter is provided.

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

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

`p_adjust_method` default "fdr"; p.adjust method; see method parameter of `p.adjust` function for available options. `p_adjust_method = "none"` can disable the p value adjustment.

`p_adjust_type` default "All"; "All", "Taxa" or "Env"; P value adjustment type. "Env": adjustment for each environmental variable separately; "Taxa": adjustment for each taxon separately; "All": adjustment for all the data together no matter whether `by_group` is provided.

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

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

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

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

`tmp_input_maaslin2` default "tmp_input"; the temporary folder used to save the input files for Maaslin2.

`tmp_output_maaslin2` default "tmp_output"; the temporary folder used to save the output files of Maaslin2.

`cor_method` deprecated. Please use `method` argument instead.

`...` parameters passed to `Maaslin2` function of `Maaslin2` package.

Returns: `res_cor` stored in the object.

Examples:

```
\donttest{
t2 <- trans_diff$new(dataset = dataset, method = "rf", group = "Group", rf_taxa_level = "Genus")
t1 <- trans_env$new(dataset = dataset, add_data = env_data_16S[, 4:11])
```

```
t1$cal_cor(use_data = "other", p_adjust_method = "fdr", other_taxa = t2$res_diff$Taxa[1:40])
}
```

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

Usage:

```
trans_env$plot_cor(
  color_vector = c("#053061", "white", "#A50026"),
  color_palette = NULL,
  filter_feature = NULL,
  filter_env = NULL,
  keep_full_name = FALSE,
  keep_prefix = TRUE,
  text_y_order = NULL,
  text_x_order = NULL,
  xtext_angle = 30,
  xtext_size = 10,
  xtext_color = "black",
  ytext_italic = FALSE,
  ytext_size = NULL,
  ytext_color = "black",
  ytext_position = "right",
  sig_label_size = 4,
  font_family = NULL,
  cluster_ggplot = "none",
  cluster_height_rows = 0.2,
  cluster_height_cols = 0.2,
  na.value = "grey50",
  trans = "identity",
  ylab_type_italic = deprecated(),
  text_y_position = deprecated(),
  ...
)
```

Arguments:

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

`color_palette` default NULL; a customized palette with more color values to be used instead of the parameter `color_vector`.

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

`filter_env` default NULL; character vector; used to filter environmental variables that only have labels in the `filter_env` vector. For example, `filter_env = ""` can be used to remove features that only have "", no any "*".

`keep_full_name` default FALSE; whether use the complete taxonomic name.

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

`text_y_order` default NULL; character vector; customized text for y axis; shown in the plot from the top down. The input should be consistent with the feature names in the `res_cor` table.

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

`xtext_angle` default 30; number ranging from 0 to 90; used to adjust x axis text angle.

`xtext_size` default 10; x axis text size.

`xtext_color` default "black"; x axis text color.

`ytext_italic` default FALSE; whether use italic for y axis text.

`ytext_size` default NULL; y axis text size. NULL means default ggplot2 value.

`ytext_color` default "black"; y axis text color.

`ytext_position` default "right"; "left" or "right"; the y axis text position.

`sig_label_size` default 4; the size of significance label shown in the cell.

`font_family` default NULL; font family used.

`cluster_ggplot` default "none"; add clustering dendrogram for `ggplot2` based heatmap. Available options: "none", "row", "col" or "both". "none": no any clustering used; "row": add clustering for rows; "col": add clustering for columns; "both": add clustering for both rows and columns.

`cluster_height_rows` default 0.2, the dendrogram plot height for rows; available when `cluster_ggplot` is not "none".

`cluster_height_cols` default 0.2, the dendrogram plot height for columns; available when `cluster_ggplot` is not "none".

`na.value` default "grey50"; the color for the missing values.

`trans` default "identity"; the transformation for continuous scales in the legend; see the `trans` item in `ggplot2::scale_colour_gradientn`.

`ylab_type_italic` deprecated. Please use `ytext_italic` argument instead.

`text_y_position` deprecated. Please use `ytext_position` argument instead.

`...` paremeters passed to `ggplot2::geom_tile`.

Returns: ggplot2 object.

Examples:

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

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

Usage:

```
trans_env$plot_scatterfit(
  x = NULL,
  y = NULL,
  group = NULL,
  group_order = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  shape_values = NULL,
  type = c("cor", "lm")[1],
  cor_method = "pearson",
```

```
  label_sep = ";",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  x_axis_title = "",
  y_axis_title = "",
  point_size = 5,
  point_alpha = 0.6,
  line_size = 0.8,
  line_color = "black",
  line_se = TRUE,
  line_se_color = "grey70",
  line_alpha = 0.5,
  pvalue_trim = 4,
  cor_coef_trim = 3,
  lm_equation = TRUE,
  lm_fir_trim = 2,
  lm_sec_trim = 2,
  lm_squ_trim = 2,
  ...
)
```

Arguments:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`point_size` default 5; point size value.

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

`line_size` default 0.8; line size value.

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

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

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

`line_alpha` default 0.5; alpha value for the color transparency of line confidence interval.

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

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

`lm_equation` default TRUE; whether include the equation in the label when `type = "lm"`.

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

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

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

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

Returns: ggplot.

Examples:

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

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

Usage:

```
trans_env$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

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

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


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


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

## Not run:
# Spearman correlation
t1$cal_autocor(upper = list(continuous = GGally::wrap("cor", method= "spearman")))

## End(Not run)

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


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


## ------------------------------------------------
## Method `trans_env$cal_ordination_anova`
## ------------------------------------------------


t1$cal_ordination_anova()


## ------------------------------------------------
## Method `trans_env$cal_ordination_envfit`
## ------------------------------------------------


t1$cal_ordination_envfit()
```

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


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


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


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


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


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


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


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


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


t1$plot_cor()
```

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


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

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

Description

This class is a wrapper for a series of functional prediction analysis on species and communities, including the prokaryotic trait prediction based on Louca et al. (2016) <doi:10.1126/science.aaf4507> and Lim et al. (2020) <10.1038/s41597-020-0516-5>, or fungal trait prediction 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$trans_spe_func_perc()`
- `trans_func$plot_spe_func_perc()`
- `trans_func$cal_tax4fun2()`
- `trans_func$cal_tax4fun2_FRI()`
- `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 unknown according to the Kingdom information. In this case, if the user still want to use the function to identify species traits, please provide "prok" or "fungi" manually, e.g. `t1$for_what <- "prok"`.

Examples:

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

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

Usage:

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

Arguments:

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

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

'NJC19' NJC19: Lim et al. (2020). Large-scale metabolic interaction network of the mouse and human gut microbiota. Scientific Data, 7(1). <10.1038/s41597-020-0516-5>. Note that the matching in this database is performed at the species level, hence utilizing it demands a higher level of precision in regards to the assignments of species in the taxonomic information table.

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

'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' version: FungalTraits_1.2_ver_16Dec_2020V.1.2; 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>

`FUNGuild_confidence` default c("Highly Probable", "Probable", "Possible"). Selected 'confidenceRanking' when `fungi_database = "FUNGuild"`.

Returns: `res_spe_func` stored in object.

Examples:

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

Method `cal_spe_func_perc()`**:** Calculating the percentages of species with specific trait in communities. The percentages of the taxa with specific trait can reflect corresponding functional

potential in the community. So this method is one representation of functional redundancy (FR) without the consideration of phylogenetic distance among taxa. The FR is defined:

$$FR_{kj}^{unweighted} = \frac{N_j}{N_k}$$

$$FR_{kj}^{weighted} = \frac{\sum_{i=1}^{N_j} A_i}{\sum_{i=1}^{N_k} A_i}$$

where FR_{kj} denotes the FR for sample k and function j. N_k is the species number in sample k. N_j is the number of species with function j in sample k. A_i is the abundance (counts) of species i in sample k.

Usage:

```
trans_func$cal_spe_func_perc(abundance_weighted = FALSE, perc = TRUE, dec = 2)
```

Arguments:

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

`perc` default TRUE; whether to use percentages in the result. If TRUE, value is bounded between 0 and 100. If FALSE, the result is relative proportion ('abundance_weighted = FALSE') or relative abundance ('abundance_weighted = TRUE') bounded between 0 and 1.

`dec` default 2; remained decimal places.

Returns: `res_spe_func_perc` stored in the object.

Examples:

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

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

Usage:

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

Arguments:

`use_func` default NULL; the function name.

Returns: None.

Examples:

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

Method `trans_spe_func_perc()`**:** Transform the `res_spe_func_perc` table to the long table format for the following visualization. Also add the group information if the database has hierarchical groups.

Usage:

```
trans_func$trans_spe_func_perc()
```

Returns: res_spe_func_perc_trans stored in the object.

Examples:

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

Method plot_spe_func_perc(): Plot the percentages of species with specific trait in communities.

Usage:

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

Arguments:

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

order_x default NULL; character vector; to sort the x axis text; can be also used to select partial samples to show.

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

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

Returns: ggplot2.

Examples:

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

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

Usage:

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

Arguments:

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

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

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

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

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

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

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

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

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

Returns: `res_tax4fun2_KO` and `res_tax4fun2_pathway` in object.

Examples:

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

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

Usage:

```
trans_func$cal_tax4fun2_FRI()
```

Returns: res_tax4fun2_aFRI and res_tax4fun2_rFRI in object.

Examples:

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

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

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

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

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


t1$cal_spe_func(prok_database = "FAPROTAX")


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


t1$cal_spe_func_perc(abundance_weighted = TRUE)


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


t1$show_prok_func(use_func = "methanotrophy")


## ------------------------------------------------
## Method `trans_func$trans_spe_func_perc`
## ------------------------------------------------


t1$trans_spe_func_perc()


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


t1$plot_spe_func_perc()
```

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

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

## End(Not run)

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

## Not run:
t1$cal_tax4fun2_FRI()

## End(Not run)
```

`trans_network` *Create* `trans_network` *object for network analysis.*

Description

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

Methods

Public methods:

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

- trans_network$cal_sum_links()
- trans_network$plot_sum_links()
- trans_network$random_network()
- trans_network$trans_comm()
- trans_network$print()
- trans_network$clone()

Method `new()`**:** Create the `trans_network` object, store the important intermediate data and calculate correlations if `cor_method` parameter is not NULL.

Usage:

```
trans_network$new(
  dataset = NULL,
  cor_method = NULL,
  use_WGCNA_pearson_spearman = FALSE,
  use_NetCoMi_pearson_spearman = FALSE,
  use_sparcc_method = c("NetCoMi", "SpiecEasi")[1],
  taxa_level = "OTU",
  filter_thres = 0,
  nThreads = 1,
  SparCC_simu_num = 100,
  env_cols = NULL,
  add_data = NULL,
  ...
)
```

Arguments:

`dataset` default NULL; the object of `microtable` class. Default NULL means customized analysis.

`cor_method` default NULL; NULL or one of "bray", "pearson", "spearman", "sparcc", "bicor", "cclasso" and "ccrepe"; All the methods refered to `NetCoMi` package are performed based on `netConstruct` function of `NetCoMi` package and require `NetCoMi` to be installed from Github (https://github.com/stefpeschel/NetCoMi); For the algorithm details, please see Peschel et al. 2020 Brief. Bioinform <doi: 10.1093/bib/bbaa290>;

NULL NULL denotes non-correlation network, i.e. do not use correlation-based network. If so, the return res_cor_p list will be NULL.

'bray' 1-B, where B is Bray-Curtis dissimilarity; based on `vegan::vegdist` function

'pearson' Pearson correlation; If `use_WGCNA_pearson_spearman` and `use_NetCoMi_pearson_spearman` are both FALSE, use the function `cor.test` in R; `use_WGCNA_pearson_spearman = TRUE` invoke `corAndPvalue` function of `WGCNA` package; `use_NetCoMi_pearson_spearman = TRUE` invoke `netConstruct` function of `NetCoMi` package

'spearman' Spearman correlation; other details are same with the 'pearson' option

'sparcc' SparCC algorithm (Friedman & Alm, PLoS Comp Biol, 2012, <doi:10.1371/journal.pcbi.1002687>); use NetCoMi package when `use_sparcc_method = "NetCoMi"`; use `SpiecEasi` package when `use_sparcc_method = "SpiecEasi"` and require `SpiecEasi` to be installed from Github (https://github.com/zdk123/SpiecEasi)

'bicor' Calculate biweight midcorrelation efficiently for matrices based on `WGCNA::bicor` function; This option can invoke `netConstruct` function of `NetCoMi` package; Make sure `WGCNA` and `NetCoMi` packages are both installed

'cclasso' Correlation inference of Composition data through Lasso method based on `netConstruct` function of `NetCoMi` package; for details, see `NetCoMi::cclasso` function

'ccrepe' Calculates compositionality-corrected p-values and q-values for compositional data using an arbitrary distance metric based on `NetCoMi::netConstruct` function; also see `NetCoMi::ccrepe` function

`use_WGCNA_pearson_spearman` default FALSE; whether use WGCNA package to calculate correlation when `cor_method` = "pearson" or "spearman".

`use_NetCoMi_pearson_spearman` default FALSE; whether use NetCoMi package to calculate correlation when `cor_method` = "pearson" or "spearman". The important difference between NetCoMi and others is the features of zero handling and data normalization; See <doi: 10.1093/bib/bbaa290>.

`use_sparcc_method` default `c("NetCoMi", "SpiecEasi")[1]`; use `NetCoMi` package or `SpiecEasi` package to perform SparCC when `cor_method` = `"sparcc"`.

`taxa_level` default "OTU"; taxonomic rank; 'OTU' denotes using feature abundance table; other available options should be one of the colnames of `tax_table` of input dataset.

`filter_thres` default 0; the relative abundance threshold.

`nThreads` default 1; the CPU thread number; available when `use_WGCNA_pearson_spearman` = `TRUE` or `use_sparcc_method` = `"SpiecEasi"`.

`SparCC_simu_num` default 100; SparCC simulation number for bootstrap when `use_sparcc_method` = `"SpiecEasi"`.

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

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

`...` parameters pass to `NetCoMi::netConstruct` for other operations, such as zero handling and/or data normalization when cor_method and other parameters refer to `NetCoMi` package.

Returns: `res_cor_p` list with the correlation (association) matrix and p value matrix. Note that when `cor_method` and other parameters refer to `NetCoMi` package, the p value table are all zero as the significant associations have been selected.

Examples:

```
\donttest{
data(dataset)
# for correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.0002)
# for non-correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = NULL)
}
```

Method `cal_network()`**:** Construct network based on the `igraph` package or `SpiecEasi` package or `julia FlashWeave` package or `beemStatic` package.

Usage:

```
trans_network$cal_network(
  network_method = c("COR", "SpiecEasi", "gcoda", "FlashWeave", "beemStatic")[1],
```

```
  COR_p_thres = 0.01,
  COR_p_adjust = "fdr",
  COR_weight = TRUE,
  COR_cut = 0.6,
  COR_optimization = FALSE,
  COR_optimization_low_high = c(0.01, 0.8),
  COR_optimization_seq = 0.01,
  SpiecEasi_method = "mb",
  FlashWeave_tempdir = NULL,
  FlashWeave_meta_data = FALSE,
  FlashWeave_other_para = "alpha=0.01,sensitive=true,heterogeneous=true",
  FlashWeave_gml = NULL,
  beemStatic_t_strength = 0.001,
  beemStatic_t_stab = 0.8,
  add_taxa_name = "Phylum",
  delete_unlinked_nodes = TRUE,
  usename_rawtaxa_notOTU = FALSE,
  ...
)
```

Arguments:

`network_method` default "COR"; "COR", "SpiecEasi", "gcoda", "FlashWeave" or "beemStatic"; `network_method = NULL` means skipping the network construction for the customized use. The option details:

- **'COR'** correlation-based network; use the correlation and p value matrices in `res_cor_p` list stored in the object; See Deng et al. (2012) <doi:10.1186/1471-2105-13-113> for other details
- **'SpiecEasi'** `SpiecEasi` network; relies on algorithms of sparse neighborhood and inverse covariance selection; belong to the category of conditional dependence and graphical models; see https://github.com/zdk123/SpiecEasi for installing the R package; see Kurtz et al. (2015) <doi:10.1371/journal.pcbi.1004226> for the algorithm details
- **'gcoda'** hypothesize the logistic normal distribution of microbiome data; use penalized maximum likelihood method to estimate the sparse structure of inverse covariance for latent normal variables to address the high dimensionality of the microbiome data; belong to the category of conditional dependence and graphical models; depend on the R `NetCoMi` package https://github.com/stefpeschel/NetCoMi; see FANG et al. (2017) <doi:10.1089/cmb.2017.0054> for the algorithm details
- **'FlashWeave'** `FlashWeave` network; Local-to-global learning framework; belong to the category of conditional dependence and graphical models; good performance on heterogenous datasets to find direct associations among taxa; see https://github.com/meringlab/FlashWeave.jl for installing `julia` language and `FlashWeave` package; julia must be in the computer system env path, otherwise the program can not find it; see Tackmann et al. (2019) <doi:10.1016/j.cels.2019.08.002> for the algorithm details
- **'beemStatic'** `beemStatic` network; extend generalized Lotka-Volterra model to cases of cross-sectional datasets to infer interaction among taxa based on expectation-maximization algorithm; see https://github.com/CSB5/BEEM-static for installing the R package; see Li et al. (2021) <doi:10.1371/journal.pcbi.1009343> for the algorithm details

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

COR_p_adjust default "fdr"; p value adjustment method, see method parameter of p.adjust function for available options, in which COR_p_adjust = "none" means giving up the p value adjustment.

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

COR_cut default 0.6; correlation coefficient threshold for the correlation network.

COR_optimization default FALSE; whether use random matrix theory (RMT) based method to determine the correlation coefficient; see https://doi.org/10.1186/1471-2105-13-113

COR_optimization_low_high default c(0.01, 0.8); the low and high value threshold used for the RMT optimization; only useful when COR_optimization = TRUE.

COR_optimization_seq default 0.01; the interval of correlation coefficient used for RMT optimization; only useful when COR_optimization = TRUE.

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

FlashWeave_tempdir default NULL; The temporary directory used to save the temporary files for running FlashWeave; If not assigned, use the system user temp.

FlashWeave_meta_data default FALSE; whether use env data for the optimization, If TRUE, the function automatically find the env_data in the object and generate a file for meta_data_path parameter of FlashWeave package.

FlashWeave_other_para default "alpha=0.01,sensitive=true,heterogeneous=true"; the parameters passed to julia FlashWeave package; user can change the parameters or add more according to FlashWeave help document; An exception is meta_data_path parameter as it is generated based on the data inside the object, see FlashWeave_meta_data parameter for the description.

FlashWeave_gml default NULL; The path of FlashWeave output gml file for customized usage. This parameter is provided for some customized needs. For instance, it can be cumbersome to input bacterial and fungal abundances as separate input files for network analysis using the above parameter. Users can run FlashWeave on their own, and then provide the resulting gml file to this parameter, which allows them to continue using other functions.

beemStatic_t_strength default 0.001; for network_method = "beemStatic"; the threshold used to limit the number of interactions (strength); same with the t.strength parameter in showInteraction function of beemStatic package.

beemStatic_t_stab default 0.8; for network_method = "beemStatic"; the threshold used to limit the number of interactions (stability); same with the t.stab parameter in showInteraction function of beemStatic package.

add_taxa_name default "Phylum"; one or more taxonomic rank name; used to add taxonomic rank name to network node properties.

delete_unlinked_nodes default TRUE; whether delete the nodes without any link.

usename_rawtaxa_notOTU default FALSE; whether use OTU name as representatives of taxa when taxa_level != "OTU". Default FALSE means using taxonomic information of taxa_level instead of OTU name.

... parameters pass to SpiecEasi::spiec.easi when network_method = "SpiecEasi"; pass to NetCoMi::netConstruct when network_method = "gcoda"; pass to beemStatic::func.EM when network_method = "beemStatic".

Returns: res_network stored in object.

Examples:

```
\dontrun{
# for correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.001)
t1$cal_network(COR_p_thres = 0.05, COR_cut = 0.6)
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.003)
t1$cal_network(network_method = "SpiecEasi", SpiecEasi_method = "mb")
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.005)
t1$cal_network(network_method = "beemStatic")
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.001)
t1$cal_network(network_method = "FlashWeave")
}
```

Method `cal_module()`**:** Calculate network modules and add module names to the network node properties.

Usage:

```
trans_network$cal_module(
  method = "cluster_fast_greedy",
  module_name_prefix = "M"
)
```

Arguments:

`method` default "cluster_fast_greedy"; the method used to find the optimal community structure of a graph; the following are available functions (options) from igraph package: `"cluster_fast_greedy"`, `"cluster_walktrap"`, `"cluster_edge_betweenness"`, `"cluster_infomap"`, `"cluster_label_prop"`, `"cluster_leading_eigen"`, `"cluster_louvain"`, `"cluster_spinglass"`, `"cluster_optimal"`.
For the details of these functions, please see the help document, such as `help(cluster_fast_greedy)`; Note that the default `"cluster_fast_greedy"` method can not be applied to directed network. If directed network is provided, the function can automatically switch the default method from `"cluster_fast_greedy"` to `"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: `res_network` with modules, stored in object.

Examples:

```
\donttest{
t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.0002)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1$cal_module(method = "cluster_fast_greedy")
}
```

Method `save_network()`**:** Save network as gexf style, which can be opened by Gephi (https://gephi.org/).

Usage:

```
trans_network$save_network(filepath = "network.gexf", ...)
```

Arguments:

`filepath` default "network.gexf"; file path to save the network.

`...` parameters pass to `gexf` function of rgexf package except for `nodes`, `edges`, `edgesLabel`, `edgesWeight`, `nodesAtt`, `edgesAtt` and `meta`.

Returns: None

Examples:

```
\dontrun{
t1$save_network(filepath = "network.gexf")
}
```

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

Usage:

```
trans_network$cal_network_attr()
```

Returns: `res_network_attr` stored in object.

Examples:

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

Method `get_node_table()`**:** Get the node property table. The properties include the node names, modules allocation, degree, betweenness, abundance, taxonomy, within-module connectivity (zi) and among-module connectivity (Pi) <doi:10.1186/1471-2105-13-113; 10.1016/j.geoderma.2022.115866>.

Usage:

```
trans_network$get_node_table(node_roles = TRUE)
```

Arguments:

`node_roles` default TRUE; whether calculate the node roles <doi:10.1038/nature03288; 10.1186/1471-2105-13-113>. The role of node i is characterized by its within-module connectivity (zi) and among-module connectivity (Pi) as follows

$$z_i = \frac{k_{ib} - \bar{k_b}}{\sigma_{k_b}}$$

$$P_i = 1 - \sum_{c=1}^{N_M} \left(\frac{k_{ic}}{k_i} \right)^2$$

where k_{ib} is the number of links of node i to other nodes in its module b, $\bar{k_b}$ and σ_{k_b} are the average and standard deviation of within-module connectivity, respectively over all the nodes in module b, k_i is the number of links of node i in the whole network, k_{ic} is the number of links from node i to nodes in module c, and N_M is the number of modules in the network.

Returns: `res_node_table` in object; Abundance expressed as a percentage; betweenness_centrality: betweenness centrality; betweenness_centrality: closeness centrality; eigenvector_centrality: eigenvector centrality; z: within-module connectivity; p: among-module connectivity.

Examples:

```
\donttest{
t1$get_node_table(node_roles = TRUE)
}
```

Method `get_edge_table()`**:** Get the edge property table, including connected nodes, label and weight.

Usage:

```
trans_network$get_edge_table()
```

Returns: `res_edge_table` in object.

Examples:

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

Method `get_adjacency_matrix()`**:** Get the adjacency matrix from the network graph.

Usage:

```
trans_network$get_adjacency_matrix(...)
```

Arguments:

`...` parameters passed to as_adjacency_matrix function of `igraph` package.

Returns: `res_adjacency_matrix` in object.

Examples:

```
\donttest{
t1$get_adjacency_matrix(attr = "weight")
}
```

Method `plot_network()`**:** Plot the network based on a series of methods from other packages, such as `igraph`, `ggraph` and `networkD3`. The networkD3 package provides dynamic network. It is especially useful for a glimpse of the whole network structure and finding the interested nodes and edges in a large network. In contrast, the igraph and ggraph methods are suitable for relatively small network.

Usage:

```
trans_network$plot_network(
  method = c("igraph", "ggraph", "networkD3")[1],
  node_label = "name",
  node_color = NULL,
  ggraph_layout = "fr",
  ggraph_node_size = 2,
  ggraph_node_text = TRUE,
  ggraph_text_color = NULL,
  ggraph_text_size = 3,
  networkD3_node_legend = TRUE,
  networkD3_zoom = TRUE,
  ...
)
```

Arguments:

`method` default "igraph"; The available options:

'igraph' call `plot.igraph` function in `igraph` package for a static network; see plot.igraph for the parameters

'ggraph' call ggraph function in ggraph package for a static network

'networkD3' use forceNetwork function in networkD3 package for a dynamic network; see forceNetwork function for the parameters

node_label default "name"; node label shown in the plot for method = "ggraph" or method = "networkD3"; Please see the column names of object$res_node_table, which is the returned table of function object$get_node_table; User can select other column names in res_node_table.

node_color default NULL; node color assignment for method = "ggraph" or method = "networkD3"; Select a column name of object$res_node_table, such as "module".

ggraph_layout default "fr"; for method = "ggraph"; see layout parameter of create_layout function in ggraph package.

ggraph_node_size default 2; for method = "ggraph"; the node size.

ggraph_node_text default TRUE; for method = "ggraph"; whether show the label text of nodes.

ggraph_text_color default NULL; for method = "ggraph"; a column name of object$res_node_table used to assign label text colors.

ggraph_text_size default 3; for method = "ggraph"; the node label text size.

networkD3_node_legend default TRUE; used for method = "networkD3"; logical value to enable node colour legends; Please see the legend parameter in networkD3::forceNetwork function.

networkD3_zoom default TRUE; used for method = "networkD3"; logical value to enable (TRUE) or disable (FALSE) zooming; Please see the zoom parameter in networkD3::forceNetwork function.

... parameters passed to plot.igraph function when method = "igraph" or forceNetwork function when method = "networkD3".

Returns: network plot.

Examples:

```
\donttest{
t1$plot_network(method = "igraph", layout = layout_with_kk)
t1$plot_network(method = "ggraph", node_color = "module")
t1$plot_network(method = "networkD3", node_color = "module")
}
```

Method `cal_eigen()`**:** Calculate eigengenes of modules, i.e. the first principal component based on PCA analysis, and the percentage of variance <doi:10.1186/1471-2105-13-113>.

Usage:

```
trans_network$cal_eigen()
```

Returns: res_eigen and res_eigen_expla in object.

Examples:

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

Method `plot_taxa_roles()`**:** Plot the roles or metrics of nodes based on the res_node_table data (coming from function get_node_table) stored in the object.

Usage:

```
trans_network$plot_taxa_roles(
  use_type = c(1, 2)[1],
  roles_color_background = FALSE,
  roles_color_values = NULL,
  add_label = FALSE,
  add_label_group = c("Network hubs", "Module hubs", "Connectors"),
  add_label_text = "name",
  label_text_size = 4,
  label_text_color = "grey50",
  label_text_italic = FALSE,
  label_text_parse = FALSE,
  plot_module = FALSE,
  x_lim = c(0, 1),
  use_level = "Phylum",
  show_value = c("z", "p"),
  show_number = 1:10,
  plot_color = "Phylum",
  plot_shape = "taxa_roles",
  plot_size = "Abundance",
  color_values = RColorBrewer::brewer.pal(12, "Paired"),
  shape_values = c(16, 17, 7, 8, 15, 18, 11, 10, 12, 13, 9, 3, 4, 0, 1, 2, 14),
  ...
)
```

Arguments:

`use_type` default 1; 1 or 2; 1 represents taxa roles plot (node roles include Module hubs, Network hubs, Connectors and Peripherals <doi:10.1038/nature03288; 10.1186/1471-2105-13-113>). The 'p' column (Pi, among-module connectivity) in `res_node_table` table is used in x-axis. The 'z' column (Zi, within-module connectivity) is used in y-axis; 2 represents the layered plot with taxa as x axis and the index (e.g., Zi and Pi) as y axis. Please refer to `res_node_table` data stored in the object for the detailed information.

`roles_color_background` default FALSE; for use_type=1; TRUE: use background colors for each area; FALSE: use classic point colors.

`roles_color_values` default NULL; for use_type=1; color palette for background or points.

`add_label` default FALSE; for use_type = 1; whether add labels for the points.

`add_label_group` default c("Network hubs", "Module hubs", "Connectors"); If `add_label = TRUE`, which part in `taxa_roles` column is used to show labels; character vectors.

`add_label_text` default "name"; If add_label = TRUE; which column of object$res_node_table is used to label the text.

`label_text_size` default 4; The text size of the label.

`label_text_color` default "grey50"; The text color of the label.

`label_text_italic` default FALSE; whether use italic style for the label text.

`label_text_parse` default FALSE; whether parse the label text. See the parse parameter in `ggrepel::geom_text_repel` function.

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

`x_lim` default c(0, 1); for use_type=1; x axis range when roles_color_background = FALSE.

use_level default "Phylum"; for use_type=2; used taxonomic level in x axis.

show_value default c("z", "p"); for use_type=2; indexes used in y axis. Please see res_node_table in the object for other available indexes.

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; variable for color.

plot_shape default "taxa_roles"; for use_type=2; variable for shape.

plot_size default "Abundance"; for use_type=2; used for point size; a fixed number (e.g. 5) is also acceptable.

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.

... parameters pass to geom_point function of ggplot2 package.

Returns: ggplot.

Examples:

```
\donttest{
t1$plot_taxa_roles(roles_color_background = FALSE)
}
```

Method `subset_network()`**:** Subset of the network.

Usage:

```
trans_network$subset_network(
  node = NULL,
  edge = NULL,
  rm_single = TRUE,
  node_alledges = FALSE,
  return_igraph = TRUE
)
```

Arguments:

node default NULL; provide the node names that will be used in the sub-network.

edge default NULL; provide the edge label or numbers that need to be remained. For the edge label, it should must be "+" or "-". For the numbers, they should fall within the range of rows in res_edge_table of the object.

rm_single default TRUE; whether remove the nodes without any edge in the sub-network. So this function can also be used to remove the nodes withou any edge when node and edge are both NULL.

node_alledges default FALSE; whether remain the nodes and edges that related to the nodes provided in node parameter. When this parameter is set to TRUE, the network will filter based on edges rather than directly on nodes. The logic is that if at least one of the two nodes connected by an edge is included in the nodes provided by the node parameter, the edge will be retained. Otherwise, it will be filtered out. When this parameter is set to FALSE, the network will filter directly based on the node parameter. Any nodes not included in the node parameter will be filtered out.

return_igraph default TRUE; whether return the network with igraph format. If FALSE, return trans_network object.

Returns: a new network

Examples:

```
\donttest{
t1$subset_network(node = t1$res_node_table %>% base::subset(module == "M1") %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1
}
```

Method `cal_powerlaw()`**:** Fit degrees to a power law distribution. First, perform a bootstrapping hypothesis test to determine whether degrees follow a power law distribution. If the distribution follows power law, then fit degrees to power law distribution and return the parameters.

Usage:

```
trans_network$cal_powerlaw(...)
```

Arguments:

`...` parameters pass to bootstrap_p function in poweRlaw package.

Returns: `res_powerlaw_p` and `res_powerlaw_fit`; see `poweRlaw::bootstrap_p` function for the bootstrapping p value details; see `igraph::fit_power_law` function for the power law fit return details.

Examples:

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

Method `cal_sum_links()`**:** This function is used to sum the links number from one taxa to another or in the same taxa, for example, at Phylum level. This is very useful to fast see how many nodes are connected between different taxa or within the taxa.

Usage:

```
trans_network$cal_sum_links(taxa_level = "Phylum")
```

Arguments:

`taxa_level` default "Phylum"; taxonomic rank.

Returns: `res_sum_links_pos` and `res_sum_links_neg` in object.

Examples:

```
\donttest{
t1$cal_sum_links(taxa_level = "Phylum")
}
```

Method `plot_sum_links()`**:** Plot the summed linkages among taxa.

Usage:

```
trans_network$plot_sum_links(
  plot_pos = TRUE,
  plot_num = NULL,
  color_values = RColorBrewer::brewer.pal(8, "Dark2"),
  method = c("chorddiag", "circlize")[1],
  ...
)
```

Arguments:

`plot_pos` default TRUE; If TRUE, plot the summed positive linkages; If FALSE, plot the summed negative linkages.

`plot_num` default NULL; number of taxa presented in the plot.

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

`method` default c("chorddiag", "circlize")[1]; chorddiag package <https://github.com/mattflor/chorddiag> or circlize package.

`...` pass to `chorddiag::chorddiag` function when `method = "chorddiag"` or `circlize::chordDiagram` function when `method = "circlize"`. Note that for `circlize::chordDiagram` function, `keep.diagonal`, `symmetric` and `self.link` parameters have been fixed to fit the input data.

Returns: please see the invoked function.

Examples:

```
\dontrun{
test1$plot_sum_links(method = "chorddiag", plot_pos = TRUE, plot_num = 10)
test1$plot_sum_links(method = "circlize", transparency = 0.2,
  annotationTrackHeight = circlize::mm_h(c(5, 5)))
}
```

Method `random_network()`**:** Generate random networks, compare them with the empirical network and get the p value of topological properties. The generation of random graph is based on the `erdos.renyi.game` function of igraph package. The numbers of vertices and edges in the random graph are same with the empirical network stored in the object.

Usage:

```
trans_network$random_network(runs = 100, output_sim = FALSE)
```

Arguments:

`runs` default 100; simulation number of random network.

`output_sim` default FALSE; whether output each simulated network result.

Returns: a data.frame with the following components:

`Observed` Topological properties of empirical network

`Mean_sim` Mean of properties of simulated networks

`SD_sim` SD of properties of simulated networks

`p_value` Significance, i.e. p values

When `output_sim = TRUE`, the columns from the five to the last are each simulated result.

Examples:

```
\dontrun{
t1$random_network(runs = 100)
}
```

Method `trans_comm()`**:** Transform classifed features to community-like microtable object for further analysis, such as module-taxa table.

Usage:

```
trans_network$trans_comm(use_col = "module", abundance = TRUE)
```

Arguments:

`use_col` default "module"; which column to use as the 'community'; must be one of the name of res_node_table from function `get_node_table`.

`abundance` default TRUE; whether sum abundance of taxa. TRUE: sum the abundance for a taxon across all samples; FALSE: sum the frequency for a taxon across all samples.

Returns: a new `microtable` class.

Examples:

```
\donttest{
t2 <- t1$trans_comm(use_col = "module")
}
```

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

Usage:

```
trans_network$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_network$new`
## ------------------------------------------------


data(dataset)
# for correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.0002)
# for non-correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = NULL)


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

## Not run:
# for correlation network
t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.001)
t1$cal_network(COR_p_thres = 0.05, COR_cut = 0.6)
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.003)
t1$cal_network(network_method = "SpiecEasi", SpiecEasi_method = "mb")
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.005)
```

```
t1$cal_network(network_method = "beemStatic")
t1 <- trans_network$new(dataset = dataset, cor_method = NULL, filter_thres = 0.001)
t1$cal_network(network_method = "FlashWeave")

## End(Not run)

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


t1 <- trans_network$new(dataset = dataset, cor_method = "pearson",
taxa_level = "OTU", filter_thres = 0.0002)
t1$cal_network(COR_p_thres = 0.01, COR_cut = 0.6)
t1$cal_module(method = "cluster_fast_greedy")


## ------------------------------------------------
## Method `trans_network$save_network`
## ------------------------------------------------

## Not run:
t1$save_network(filepath = "network.gexf")

## End(Not run)

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


t1$cal_network_attr()


## ------------------------------------------------
## Method `trans_network$get_node_table`
## ------------------------------------------------


t1$get_node_table(node_roles = TRUE)


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


t1$get_edge_table()


## ------------------------------------------------
## Method `trans_network$get_adjacency_matrix`
## ------------------------------------------------
```

```
t1$get_adjacency_matrix(attr = "weight")


## ------------------------------------------------
## Method `trans_network$plot_network`
## ------------------------------------------------


t1$plot_network(method = "igraph", layout = layout_with_kk)
t1$plot_network(method = "ggraph", node_color = "module")
t1$plot_network(method = "networkD3", node_color = "module")


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


t1$cal_eigen()


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


t1$plot_taxa_roles(roles_color_background = FALSE)


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


t1$subset_network(node = t1$res_node_table %>% base::subset(module == "M1") %>%
  rownames, rm_single = TRUE)
# return a sub network that contains all nodes of module M1


## ------------------------------------------------
## Method `trans_network$cal_powerlaw`
## ------------------------------------------------


t1$cal_powerlaw()


## ------------------------------------------------
## Method `trans_network$cal_sum_links`
## ------------------------------------------------
```

```
t1$cal_sum_links(taxa_level = "Phylum")


## ------------------------------------------------
## Method `trans_network$plot_sum_links`
## ------------------------------------------------

## Not run:
test1$plot_sum_links(method = "chorddiag", plot_pos = TRUE, plot_num = 10)
test1$plot_sum_links(method = "circlize", transparency = 0.2,
  annotationTrackHeight = circlize::mm_h(c(5, 5)))

## End(Not run)

## ------------------------------------------------
## Method `trans_network$random_network`
## ------------------------------------------------

## Not run:
t1$random_network(runs = 100)

## End(Not run)

## ------------------------------------------------
## Method `trans_network$trans_comm`
## ------------------------------------------------


t2 <- t1$trans_comm(use_col = "module")
```

`trans_norm` *Feature abundance normalization/transformation.*

Description

Feature abundance normalization/transformation for a microtable object or data.frame object.

Methods

Public methods:

- `trans_norm$new()`
- `trans_norm$norm()`
- `trans_norm$clone()`

Method `new()`**:** Get a transposed abundance table if the input is microtable object. In the table, rows are samples, and columns are features. This can make the further operations same with the traditional ecological methods.

Usage:

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

Arguments:

`dataset` the `microtable` object or `data.frame` object. If it is `data.frame` object, please make sure that rows are samples, and columns are features.

Returns: data_table, stored in the object.

Examples:

```
library(microeco)
data(dataset)
t1 <- trans_norm$new(dataset = dataset)
```

Method `norm()`**:** Normalization/transformation methods.

Usage:

```
trans_norm$norm(
  method = "rarefy",
  sample.size = NULL,
  rngseed = 123,
  replace = TRUE,
  pseudocount = 1,
  intersect.no = 10,
  ct.min = 1,
  condition = NULL,
  MARGIN = NULL,
  logbase = 2,
  ...
)
```

Arguments:

`method` default "rarefy"; See the following available options.

Methods for normalization:

- `"rarefy"`: classic rarefaction based on R sample function.
- `"SRS"`: scaling with ranked subsampling method based on the SRS package provided by Lukas Beule and Petr Karlovsky (2020) <doi:10.7717/peerj.9593>.
- `"clr"`: Centered log-ratio normalization <ISBN:978-0-412-28060-3> <doi: 10.3389/fmicb.2017.02224>. It is defined:

$$clr_{ki} = \log \frac{x_{ki}}{g(x_i)}$$

where x_{ki} is the abundance of kth feature in sample i, $g(x_i)$ is the geometric mean of abundances for sample i. A pseudocount need to be added to deal with the zero. For more information, please see the 'clr' method in `decostand` function of vegan package.

- `"rclr"`: Robust centered log-ratio normalization <doi:10.1128/msystems.00016-19>. It is defined:

$$rclr_{ki} = \log \frac{x_{ki}}{g(x_i > 0)}$$

where x_{ki} is the abundance of kth feature in sample i, $g(x_i > 0)$ is the geometric mean of abundances (> 0) for sample i. In rclr, zero values are kept as zeroes, and not taken into account.

- "GMPR": Geometric mean of pairwise ratios <doi: 10.7717/peerj.4600>. For a given sample i, the size factor s_i is defined:

$$s_i = \left(\prod_{j=1}^{n} Median_{k|c_{ki}c_{kj} \neq 0} \{ \frac{c_{ki}}{c_{kj}} \} \right)^{1/n}$$

  where k denotes all the features, and n denotes all the samples. For sample i, $GMPR = \frac{x_i}{s_i}$, where x_i is the feature abundances of sample i.
- "CSS": Cumulative sum scaling normalization based on the metagenomeSeq package <doi:10.1038/nmeth.2658>. For a given sample j, the scaling factor s_j^l is defined:

$$s_j^l = \sum_{i|c_{ij} \leqslant q_j^l} c_{ij}$$

  where q_j^l is the lth quantile of sample j, that is, in sample j there are l features with counts smaller than q_j^l. c_{ij} denotes the count (abundance) of feature i in sample j. For l = 0.95m (feature number), q_j^l corresponds to the 95th percentile of the count distribution for sample j. Normalized counts $\tilde{c_{ij}} = (\frac{c_{ij}}{s_j^l})(N)$, where N is an appropriately chosen normalization constant.
- "TSS": Total sum scaling. Abundance is divided by the sequencing depth. For a given sample j, normalized counts is defined:

$$\tilde{c_{ij}} = \frac{c_{ij}}{\sum_{i=1}^{N_j} c_{ij}}$$

  where c_{ij} is the counts of feature i in sample j, and N_j is the feature number of sample j.
- "eBay": Empirical Bayes approach to normalization <10.1186/s12859-020-03552-z>. The implemented method is not tree-related. In the output, the sum of each sample is 1.
- "TMM": Trimmed mean of M-values method based on the normLibSizes function of edgeR package <doi: 10.1186/gb-2010-11-3-r25>.
- "DESeq2": Median ratio of gene counts relative to geometric mean per gene based on the DESeq function of DESeq2 package <doi: 10.1186/s13059-014-0550-8>. This option can invoke the trans_diff class and extract the normalized data from the original result. Note that either group or formula should be provided. The scaling factor is defined:

$$s_j = Median_i \frac{c_{ij}}{\left(\prod_{j=1}^{n} c_{ij} \right)^{1/n}}$$

  where c_{ij} is the counts of feature i in sample j, and n is the total sample number.
- "Wrench": Group-wise and sample-wise compositional bias factor <doi: 10.1186/s12864-018-5160-5>. Note that condition parameter is necesary to be passed to condition parameter in wrench function of Wrench package. As the input data must be microtable object, so the input condition parameter can be a column name of sample_table. The scaling factor is defined:

$$s_j = \frac{1}{p} \sum_{ij} W_{ij} \frac{X_{ij}}{\overline{X_i}}$$

  where X_{ij} represents the relative abundance (proportion) for feature i in sample j, $\overline{X_i}$ is the average proportion of feature i across the dataset, W_{ij} represents a weight specific to each technique, and p is the feature number in sample.

- "RLE": Relative log expression.

Methods based on decostand function of vegan package:

- "total": divide by margin total (default MARGIN = 1, i.e. rows - samples).
- "max": divide by margin maximum (default MARGIN = 2, i.e. columns - features).
- "normalize": make margin sum of squares equal to one (default MARGIN = 1).
- "range": standardize values into range 0...1 (default MARGIN = 2). If all values are constant, they will be transformed to 0.
- "standardize": scale x to zero mean and unit variance (default MARGIN = 2).
- "pa": scale x to presence/absence scale (0/1).
- "log": logarithmic transformation.

Other methods for transformation:

- "AST": Arc sine square root transformation.

sample.size default NULL; libray size for rarefaction when method = "rarefy" or "SRS". If not provided, use the minimum number across all samples. For "SRS" method, this parameter is passed to Cmin parameter of SRS function of SRS package.

rngseed default 123; random seed. Available when method = "rarefy" or "SRS".

replace default TRUE; see sample for the random sampling; Available when method = "rarefy".

pseudocount default 1; add pseudocount for those features with 0 abundance when method = "clr".

intersect.no default 10; the intersecting taxa number between paired sample for method = "GMPR".

ct.min default 1; the minimum number of counts required to calculate ratios for method = "GMPR".

condition default NULL; Only available when method = "Wrench". This parameter is passed to the condition parameter of wrench function in Wrench package It must be a column name of sample_table or a vector with same length of samples.

MARGIN default NULL; 1 = samples, and 2 = features of abundance table; only available when method comes from decostand function of vegan package. If MARGIN is NULL, use the default value in decostand function.

logbase default 2; The logarithm base.

... parameters pass to vegan::decostand, or metagenomeSeq::cumNorm when method = "CSS", or edgeR::normLibSizes when method = "TMM" or "RLE", or trans_diff class when method = "DESeq2", or wrench function of Wrench package when method = "Wrench".

Returns: new microtable object or data.frame object.

Examples:

```
newdataset <- t1$norm(method = "clr")
newdataset <- t1$norm(method = "log")
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_norm$new`
## ------------------------------------------------

library(microeco)
data(dataset)
t1 <- trans_norm$new(dataset = dataset)

## ------------------------------------------------
## Method `trans_norm$norm`
## ------------------------------------------------

newdataset <- t1$norm(method = "clr")
newdataset <- t1$norm(method = "log")
```

`trans_nullmodel` *Create* `trans_nullmodel` *object for null model related analysis.*

Description

This class is a wrapper for a series of null model related approaches, including the mantel correlogram analysis of phylogenetic signal, beta nearest taxon index (betaNTI), beta net relatedness index (betaNRI), NTI, NRI and RCbray (Raup–Crick Bray–Curtis) calculations. See <doi:10.1111/j.1600-0587.2010.06548.x; 10.1890/ES10-00117.1; 10.1038/ismej.2013.93; 10.1038/s41598-017-17736-w> for the algorithms and applications.

Methods

Public methods:

- `trans_nullmodel$new()`
- `trans_nullmodel$cal_mantel_corr()`
- `trans_nullmodel$plot_mantel_corr()`
- `trans_nullmodel$cal_betampd()`
- `trans_nullmodel$cal_betamntd()`
- `trans_nullmodel$cal_ses_betampd()`
- `trans_nullmodel$cal_ses_betamntd()`
- `trans_nullmodel$cal_rcbray()`
- `trans_nullmodel$cal_process()`
- `trans_nullmodel$cal_NRI()`
- `trans_nullmodel$cal_NTI()`
- `trans_nullmodel$cal_Cscore()`
- `trans_nullmodel$cal_NST()`
- `trans_nullmodel$cal_NST_test()`
- `trans_nullmodel$cal_NST_convert()`

- trans_nullmodel$clone()

Method `new()`**:**

Usage:

```
trans_nullmodel$new(
  dataset = NULL,
  filter_thres = 0,
  taxa_number = NULL,
  group = NULL,
  select_group = NULL,
  env_cols = NULL,
  add_data = NULL,
  complete_na = FALSE
)
```

Arguments:

`dataset` the object of `microtable` Class.

`filter_thres` default 0; the relative abundance threshold.

`taxa_number` default NULL; how many taxa the user want to keep, if provided, filter_thres parameter will be forcible invalid.

`group` default NULL; which column name in sample_table is selected as the group for the following selection.

`select_group` default NULL; one or more elements in `group`, used to select samples.

`env_cols` default NULL; number or name vector to select the environmental data in dataset$sample_table.

`add_data` default NULL; provide environmental data table additionally.

`complete_na` default FALSE; whether fill the NA in environmental data based on the method in mice package.

Returns: `data_comm` and `data_tree` in object.

Examples:

```
data(dataset)
data(env_data_16S)
t1 <- trans_nullmodel$new(dataset, filter_thres = 0.0005, add_data = env_data_16S)
```

Method `cal_mantel_corr()`**:** Calculate mantel correlogram.

Usage:

```
trans_nullmodel$cal_mantel_corr(
  use_env = NULL,
  break.pts = seq(0, 1, 0.02),
  cutoff = FALSE,
  ...
)
```

Arguments:

`use_env` default NULL; numeric or character vector to select env_data; if provide multiple variables or NULL, use PCA (principal component analysis) to reduce dimensionality.

`break.pts` default seq(0, 1, 0.02); see break.pts parameter in `mantel.correlog` of `vegan` package.

cutoff default FALSE; see cutoff parameter in mantel.correlog.

... parameters pass to mantel.correlog function in vegan package.

Returns: res_mantel_corr in object.

Examples:

```
\dontrun{
t1$cal_mantel_corr(use_env = "pH")
}
```

Method plot_mantel_corr(): Plot mantel correlogram.

Usage:

```
trans_nullmodel$plot_mantel_corr(point_shape = 22, point_size = 3)
```

Arguments:

point_shape default 22; the number for selecting point shape type; see ggplot2 manual for the number meaning.

point_size default 3; the point size.

Returns: ggplot.

Examples:

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

Method cal_betampd(): Calculate betaMPD (mean pairwise distance). Same with picante::comdist function, but faster.

Usage:

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

Arguments:

abundance.weighted default TRUE; whether use abundance-weighted method.

Returns: res_betampd in object.

Examples:

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

Method cal_betamntd(): Calculate betaMNTD (mean nearest taxon distance). Same with picante::comdistnt function, but faster.

Usage:

```
trans_nullmodel$cal_betamntd(
  abundance.weighted = TRUE,
  exclude.conspecifics = FALSE,
  use_iCAMP = FALSE,
  use_iCAMP_force = TRUE,
  iCAMP_tempdir = NULL,
  ...
)
```

Arguments:

`abundance.weighted` default TRUE; whether use abundance-weighted method.

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

`use_iCAMP` default FALSE; whether use `bmntd.big` function of `iCAMP` package to calculate betaMNTD. This method can store the phylogenetic distance matrix on the disk to lower the memory spending and perform the calculation parallelly.

`use_iCAMP_force` default FALSE; whether use `bmntd.big` function of `iCAMP` package automatically when the feature number is large.

`iCAMP_tempdir` default NULL; the temporary directory used to place the large tree file; If NULL; use the system user tempdir.

`...` paremeters pass to `iCAMP::pdist.big` function.

Returns: res_betamntd in object.

Examples:

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

Method `cal_ses_betampd()`**:** Calculate standardized effect size of betaMPD, i.e. beta net relatedness index (betaNRI).

Usage:

```
trans_nullmodel$cal_ses_betampd(
  runs = 1000,
 null.model = c("taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool",
    "independentswap", "trialswap")[1],
  abundance.weighted = TRUE,
  iterations = 1000
)
```

Arguments:

`runs` default 1000; simulation runs.

`null.model` default "taxa.labels"; The available options include "taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool", "independentswap"and "trialswap"; see `null.model` parameter of `ses.mntd` function in `picante` package for the algorithm details.

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

`iterations` default 1000; iteration number for part null models to perform; see iterations parameter of `picante::randomizeMatrix` function.

Returns: res_ses_betampd in object.

Examples:

```
\dontrun{
# only run 50 times for the example; default 1000
t1$cal_ses_betampd(runs = 50, abundance.weighted = TRUE)
}
```

Method `cal_ses_betamntd()`**:** Calculate standardized effect size of betaMNTD, i.e. beta nearest taxon index (betaNTI).

Usage:

```
trans_nullmodel$cal_ses_betamntd(
  runs = 1000,
 null.model = c("taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool",
    "independentswap", "trialswap")[1],
  abundance.weighted = TRUE,
  exclude.conspecifics = FALSE,
  use_iCAMP = FALSE,
  use_iCAMP_force = TRUE,
  iCAMP_tempdir = NULL,
  nworker = 2,
  iterations = 1000
)
```

Arguments:

`runs` default 1000; simulation number of null model.

`null.model` default "taxa.labels"; The available options include "taxa.labels", "richness", "frequency", "sample.pool", "phylogeny.pool", "independentswap"and "trialswap"; see `null.model` parameter of `ses.mntd` function in `picante` package for the algorithm details.

`abundance.weighted` default TRUE; whether use abundance-weighted method.

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

`use_iCAMP` default FALSE; whether use bmntd.big function of iCAMP package to calculate betaMNTD. This method can store the phylogenetic distance matrix on the disk to lower the memory spending and perform the calculation parallelly.

`use_iCAMP_force` default FALSE; whether to make use_iCAMP to be TRUE when the feature number is large.

`iCAMP_tempdir` default NULL; the temporary directory used to place the large tree file; If NULL; use the system user tempdir.

`nworker` default 2; the CPU thread number.

`iterations` default 1000; iteration number for part null models to perform; see iterations parameter of `picante::randomizeMatrix` function.

Returns: res_ses_betamntd in object.

Examples:

```
\dontrun{
# only run 50 times for the example; default 1000
t1$cal_ses_betamntd(runs = 50, abundance.weighted = TRUE, exclude.conspecifics = FALSE)
}
```

Method `cal_rcbray()`**:** Calculate Bray–Curtis-based Raup–Crick (RCbray) <doi: 10.1890/ES10-00117.1>.

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:

```
\dontrun{
# only run 50 times for the example; default 1000
t1$cal_rcbray(runs = 50)
}
```

Method `cal_process()`**:** Infer the ecological processes according to ses.betaMNTD (betaNTI)/ses.betaMPD (betaNRI) and rcbray.

Usage:

```
trans_nullmodel$cal_process(use_betamntd = TRUE, group = NULL)
```

Arguments:

`use_betamntd` default TRUE; whether use ses.betaMNTD (betaNTI); if False, use ses.betaMPD (betaNRI).

`group` default NULL; a column name in sample_table of microtable object. If provided, the analysis will be performed for each group instead of the whole.

Returns: res_process in object.

Examples:

```
\dontrun{
t1$cal_process(use_betamntd = TRUE)
}
```

Method `cal_NRI()`**:** Calculates Nearest Relative Index (NRI), equivalent to -1 times the standardized effect size of MPD.

Usage:

```
trans_nullmodel$cal_NRI(
  null.model = "taxa.labels",
  abundance.weighted = FALSE,
  runs = 999,
  ...
)
```

Arguments:

`null.model` default "taxa.labels"; Null model to use; see `null.model` parameter in `ses.mpd` function of `picante` package for available options.

`abundance.weighted` default FALSE; Should mean nearest relative distances for each species be weighted by species abundance?

`runs` default 999; Number of randomizations.

`...` paremeters pass to ses.mpd function in picante package.

Returns: res_NRI in object, equivalent to -1 times ses.mpd.

Examples:

```
\donttest{
# only run 50 times for the example; default 999
t1$cal_NRI(null.model = "taxa.labels", abundance.weighted = FALSE, runs = 50)
}
```

Method `cal_NTI()`**:** Calculates Nearest Taxon Index (NTI), equivalent to -1 times the standardized effect size of MNTD.

Usage:

```
trans_nullmodel$cal_NTI(
  null.model = "taxa.labels",
  abundance.weighted = FALSE,
  runs = 999,
  ...
)
```

Arguments:

`null.model` default "taxa.labels"; Null model to use; see `null.model` parameter in `ses.mntd` function of `picante` package for available options.

`abundance.weighted` default FALSE; Should mean nearest taxon distances for each species be weighted by species abundance?

`runs` default 999; Number of randomizations.

`...` paremeters pass to `ses.mntd` function in `picante` package.

Returns: res_NTI in object, equivalent to -1 times ses.mntd.

Examples:

```
\donttest{
# only run 50 times for the example; default 999
t1$cal_NTI(null.model = "taxa.labels", abundance.weighted = TRUE, runs = 50)
}
```

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 `bipartite::C.score` function.

Returns: vector.

Examples:

```
\dontrun{
t1$cal_Cscore(normalise = FALSE)
t1$cal_Cscore(by_group = "Group", normalise = FALSE)
}
```

Method `cal_NST()`**:** Calculate normalized stochasticity ratio (NST) based on the `NST` package.

Usage:

```
trans_nullmodel$cal_NST(method = "tNST", group, ...)
```

Arguments:

`method` default "tNST"; `'tNST'` or `'pNST'`. See the help document of `tNST` or `pNST` function in `NST` package for more details.

`group` a colname of `sample_table` in microtable object; the function can select the data from sample_table to generate a one-column (n x 1) matrix and provide it to the group parameter of `tNST` or `pNST` function.

`...` paremeters pass to `NST::tNST` or `NST::pNST` function; see the document of corresponding function for more details.

Returns: res_NST stored in the object.

Examples:

```
\dontrun{
t1$cal_NST(group = "Group", dist.method = "bray", output.rand = TRUE, SES = TRUE)
}
```

Method `cal_NST_test()`**:** Test the significance of NST difference between each pair of groups.

Usage:

```
trans_nullmodel$cal_NST_test(method = "nst.boot", ...)
```

Arguments:

`method` default "nst.boot"; "nst.boot" or "nst.panova"; see `NST::nst.boot` function or `NST::nst.panova` function for the details.

`...` paremeters pass to NST::nst.boot when method = "nst.boot" or NST::nst.panova when method = "nst.panova".

Returns: list. See the Return part of `NST::nst.boot` function or `NST::nst.panova` function in NST package.

Examples:

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

Method `cal_NST_convert()`**:** Convert NST paired long format table to symmetric matrix form.

Usage:

```
trans_nullmodel$cal_NST_convert(column = 10)
```

Arguments:

`column` default 10; which column is selected for the conversion. See the columns of `res_NST$index.pair` stored in the object.

Returns: symmetric matrix.

Examples:

```
\dontrun{
t1$cal_NST_convert(column = 10)
}
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_nullmodel$new`
## ------------------------------------------------

data(dataset)
data(env_data_16S)
t1 <- trans_nullmodel$new(dataset, filter_thres = 0.0005, add_data = env_data_16S)

## ------------------------------------------------
## Method `trans_nullmodel$cal_mantel_corr`
## ------------------------------------------------

## Not run:
t1$cal_mantel_corr(use_env = "pH")

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$plot_mantel_corr`
## ------------------------------------------------

## Not run:
t1$plot_mantel_corr()

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$cal_betampd`
## ------------------------------------------------


t1$cal_betampd(abundance.weighted = TRUE)


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


t1$cal_betamntd(abundance.weighted = TRUE)


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

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

## Not run:
# only run 50 times for the example; default 1000
t1$cal_ses_betampd(runs = 50, abundance.weighted = TRUE)

## End(Not run)

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

## Not run:
# only run 50 times for the example; default 1000
t1$cal_ses_betamntd(runs = 50, abundance.weighted = TRUE, exclude.conspecifics = FALSE)

## End(Not run)

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

## Not run:
# only run 50 times for the example; default 1000
t1$cal_rcbray(runs = 50)

## End(Not run)

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

## Not run:
t1$cal_process(use_betamntd = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$cal_NRI`
## ------------------------------------------------


# only run 50 times for the example; default 999
t1$cal_NRI(null.model = "taxa.labels", abundance.weighted = FALSE, runs = 50)


## ------------------------------------------------
## Method `trans_nullmodel$cal_NTI`
## ------------------------------------------------


# only run 50 times for the example; default 999
```

```
t1$cal_NTI(null.model = "taxa.labels", abundance.weighted = TRUE, runs = 50)


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

## Not run:
t1$cal_Cscore(normalise = FALSE)
t1$cal_Cscore(by_group = "Group", normalise = FALSE)

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$cal_NST`
## ------------------------------------------------

## Not run:
t1$cal_NST(group = "Group", dist.method = "bray", output.rand = TRUE, SES = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$cal_NST_test`
## ------------------------------------------------

## Not run:
t1$cal_NST_test()

## End(Not run)

## ------------------------------------------------
## Method `trans_nullmodel$cal_NST_convert`
## ------------------------------------------------

## Not run:
t1$cal_NST_convert(column = 10)

## End(Not run)
```

`trans_venn` *Create* `trans_venn` *object for the Venn diagram, petal plot and UpSet plot.*

Description

This class is a wrapper for a series of intersection analysis related methods, including 2- to 5-way venn diagram, more than 5-way petal or UpSet plot and intersection 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$plot_bar()
- trans_venn$trans_comm()
- trans_venn$print()
- trans_venn$clone()

Method new():

Usage:

```
trans_venn$new(dataset, ratio = NULL, name_joint = "&")
```

Arguments:

dataset the object of microtable class or a matrix-like table (data.frame or matrix object). If dataset is a matrix-like table, features must be rows.

ratio default NULL; NULL, "numratio" or "seqratio"; "numratio": calculate the percentage of feature number; "seqratio": calculate the percentage of feature abundance; NULL: no additional percentage.

name_joint default "&"; the joint mark for generating multi-sample names.

Returns: data_details and data_summary stored in the object.

Examples:

```
\donttest{
data(dataset)
t1 <- dataset$merge_samples("Group")
t1 <- trans_venn$new(dataset = t1, ratio = "numratio")
}
```

Method plot_venn(): Plot venn diagram.

Usage:

```
trans_venn$plot_venn(
  color_circle = RColorBrewer::brewer.pal(8, "Dark2"),
  fill_color = TRUE,
  text_size = 4.5,
  text_name_size = 6,
  text_name_position = NULL,
  alpha = 0.3,
  linesize = 1.1,
  petal_plot = FALSE,
  petal_color = "#BEAED4",
  petal_color_center = "#BEBADA",
  petal_a = 4,
  petal_r = 1,
  petal_use_lim = c(-12, 12),
  petal_center_size = 40,
```

```
  petal_move_xy = 4,
  petal_move_k = 2.3,
  petal_move_k_count = 1.3,
  petal_text_move = 40,
  other_text_show = NULL,
  other_text_position = c(2, 2),
  other_text_size = 5
)
```

Arguments:

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

`fill_color` default TRUE; whether fill the area color.

`text_size` default 4.5; text size in plot.

`text_name_size` default 6; name size in plot.

`text_name_position` default NULL; name position in plot.

`alpha` default .3; alpha for transparency.

`linesize` default 1.1; cycle line size.

`petal_plot` default FALSE; whether use petal plot.

`petal_color` default "#BEAED4"; color of the petals; If petal_color only has one color value, all the petals will be assigned with this color value. If petal_color has multiple colors, and the number of color values is smaller than the petal number, the function can append more colors automatically with the color interpolation.

`petal_color_center` default "#BEBADA"; color of the center in the petal plot.

`petal_a` default 4; the length of the ellipse.

`petal_r` default 1; scaling up the size of the ellipse.

`petal_use_lim` default c(-12, 12); the width of the plot.

`petal_center_size` default 40; petal center circle size.

`petal_move_xy` default 4; the distance of text to circle.

`petal_move_k` default 2.3; the distance of title to circle.

`petal_move_k_count` default 1.3; the distance of data text to circle.

`petal_text_move` default 40; the distance between two data text.

`other_text_show` default NULL; other characters used to show in the plot.

`other_text_position` default c(1, 1); the text position for text in `other_text_show`.

`other_text_size` default 5; the text size for text in `other_text_show`.

Returns: ggplot.

Examples:

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

Method `plot_bar()`**:** Plot the intersections using histogram, i.e. UpSet plot. Especially useful when samples > 5.

Usage:

```
trans_venn$plot_bar(
  left_plot = TRUE,
  sort_samples = FALSE,
  up_y_title = "Intersection size",
  up_y_title_size = 15,
  up_y_text_size = 8,
  up_bar_fill = "grey70",
  up_bar_width = 0.9,
  bottom_y_text_size = 12,
  bottom_height = 1,
  bottom_point_size = 3,
  bottom_point_color = "black",
  bottom_background_fill = "grey95",
  bottom_background_alpha = 1,
  bottom_line_width = 0.5,
  bottom_line_colour = "black",
  left_width = 0.3,
  left_bar_fill = "grey70",
  left_bar_alpha = 1,
  left_bar_width = 0.9,
  left_x_text_size = 10,
  left_background_fill = "white",
  left_background_alpha = 1
)
```

Arguments:

`left_plot` default TRUE; whether add the left bar plot to show the feature number of each sample.

`sort_samples` default FALSE; `TRUE` is used to sort samples according to the number of features in each sample. `FALSE` means the sample order is same with that in sample_table of the raw dataset.

`up_y_title` default "Intersection set"; y axis title of upper plot.

`up_y_title_size` default 15; y axis title size of upper plot.

`up_y_text_size` default 4; y axis text size of upper plot.

`up_bar_fill` default "grey70"; bar fill color of upper plot.

`up_bar_width` default 0.9; bar width of upper plot.

`bottom_y_text_size` default 12; y axis text size, i.e. sample name size, of bottom sample plot.

`bottom_height` default 1; bottom plot height relative to the upper bar plot. 1 represents the height of bottom plot is same with the upper bar plot.

`bottom_point_size` default 3; point size of bottom plot.

`bottom_point_color` default "black"; point color of bottom plot.

`bottom_background_fill` default "grey95"; fill color for the striped background in the bottom sample plot. If the parameter length is 1, use "white" to distinguish the color stripes. If the parameter length is greater than 1, use all provided colors.

`bottom_background_alpha` default 1; the color transparency for the parameter `bottom_background_fill`.

`bottom_line_width` default 0.5; the line width in the bottom plot.

`bottom_line_colour` default "black"; the line color in the bottom plot.

`left_width` default 0.3; left bar plot width relative to the right bottom plot.

`left_bar_fill` default "grey70"; fill color for the left bar plot presenting feature number.

`left_bar_alpha` default 1; the color transparency for the parameter `left_bar_fill`.

`left_bar_width` default 0.9; bar width of left plot.

`left_x_text_size` default 10; x axis text size of the left bar plot.

`left_background_fill` default "white"; fill color for the striped background in the left plot. If the parameter length is 1, use "white" to distinguish the color stripes. If the parameter length is greater than 1, use all provided colors.

`left_background_alpha` default 1; the color transparency for the parameter `left_background_fill`.

Returns: a ggplot2 object.

Examples:

```
\donttest{
t2 <- t1$plot_bar()
}
```

Method `trans_comm()`**:** Transform intersection result to community-like microtable object for further composition analysis.

Usage:

```
trans_venn$trans_comm(use_frequency = TRUE)
```

Arguments:

`use_frequency` default TRUE; whether only use OTUs occurrence frequency, i.e. presence/absence data; if FALSE, use abundance data.

Returns: a new `microtable` class.

Examples:

```
\donttest{
t2 <- t1$trans_comm(use_frequency = TRUE)
}
```

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

Usage:

```
trans_venn$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `trans_venn$new`
## ------------------------------------------------


data(dataset)
t1 <- dataset$merge_samples("Group")
t1 <- trans_venn$new(dataset = t1, ratio = "numratio")


## ------------------------------------------------
## Method `trans_venn$plot_venn`
## ------------------------------------------------


t1$plot_venn()


## ------------------------------------------------
## Method `trans_venn$plot_bar`
## ------------------------------------------------


t2 <- t1$plot_bar()


## ------------------------------------------------
## Method `trans_venn$trans_comm`
## ------------------------------------------------


t2 <- t1$trans_comm(use_frequency = TRUE)
```

Index

∗ **Description**
    microeco, 6
∗ **R6**
    dataset, 3
∗ **data.frame**
    env_data_16S, 5
    fungi_func_FungalTraits, 5
    fungi_func_FUNGuild, 5
    otu_table_16S, 18
    otu_table_ITS, 19
    phylo_tree_16S, 19
    prok_func_FAPROTAX, 19
    prok_func_NJC19_list, 20
    sample_info_16S, 20
    sample_info_ITS, 20
    taxonomy_table_16S, 21
    taxonomy_table_ITS, 21
∗ **list**
    Tax4Fun2_KEGG, 20
∗ **object**
    dataset, 3

aov, *34*

clone, 2

data.frame, *22*
dataset, 3
dropallfactors, 4

env_data_16S, 5

fungi_func_FungalTraits, 5
fungi_func_FUNGuild, 5

grepl, *13*

microeco, 6
microtable, *6*, 6, *23*, *33*, *40*, *50*, *61*, *71*, *87*, *93*, *105*, *109*, *113*, *123*, *126*

otu_table_16S, 18
otu_table_ITS, 19

phylo_tree_16S, 19
prok_func_FAPROTAX, 19
prok_func_NJC19_list, 20

sample, *111*
sample_info_16S, 20
sample_info_ITS, 20

Tax4Fun2_KEGG, 20
taxonomy_table_16S, 21
taxonomy_table_ITS, 21
tidy_taxonomy, 21
trans_abund, *6*, 22
trans_alpha, *6*, 33, *46, 47*, *65*, *72*
trans_beta, *6*, 39
trans_classifier, *6*, 49
trans_diff, *6*, 60
trans_env, *6*, *38*, *67*, 70
trans_func, *6*, 86
trans_network, *6*, 92
trans_norm, *6*, *9*, *63*, 108
trans_nullmodel, *6*, 112
trans_venn, *6*, 122

write.table, *12–14*