Package 'LegATo'

November 5, 2024

Title LegATo: Longitudinal mEtaGenomic Analysis Toolkit

Version 1.0.0

Description LegATo is a suite of open-source software tools for longitudinal microbiome analysis. It is extendable to several different study forms with optimal ease-of-use for researchers. Microbiome time-series data presents distinct challenges including complex covariate dependencies and variety of longitudinal study designs. This toolkit will allow researchers to determine which microbial taxa are affected over time by perturbations such as onset of disease or lifestyle choices, and to predict the effects of these perturbations over time, including changes in composition or stability of commensal bacteria.

License MIT + file LICENSE

URL https://wejlab.github.io/LegATo-docs/

BugReports https://github.com/wejlab/LegATo/issues

Depends R (>= 4.4.0)

Imports animalcules, data.table, dplyr, ggplot2, magrittr, MultiAssayExperiment, plyr, rlang, S4Vectors, stringr, SummarizedExperiment, tibble, tidyr, tidyselect

Suggests BiocStyle, broom, broom.mixed, circlize, ComplexHeatmap, emmeans, geepack, ggalluvial, ggeffects, grDevices, knitr, lme4, lmerTest, methods, RColorBrewer, rmarkdown, spelling, TBSignatureProfiler, testthat (>= 3.0.0), TreeSummarizedExperiment, usethis, vegan

VignetteBuilder knitr

BiocType Software

biocViews MicrobiomeData, ReproducibleResearch, SequencingData

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData FALSE

RoxygenNote 7.3.1

git_url https://git.bioconductor.org/packages/LegATo

git_branch RELEASE_3_20

git_last_commit a6dcb95

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-11-05

Author Aubrey Odom [aut, cre] (<https://orcid.org/0000-0001-7113-7598>),
Yilong Zhang [ctb] (Author of NMIT functions),
Jared Pincus [csl] (<https://orcid.org/0000-0001-6708-5262>, Naming consultant of package),
Jordan Pincus [art] (Artist of LegATo icon)

Maintainer Aubrey Odom <aodom@bu.edu>

Contents

`LegATo-package` *LegATo: LegATo: Longitudinal mEtaGenomic Analysis Toolkit*

Description

LegATo is a suite of open-source software tools for longitudinal microbiome analysis. It is extendable to several different study forms with optimal ease-of-use for researchers. Microbiome time-series data presents distinct challenges including complex covariate dependencies and variety of longitudinal study designs. This toolkit will allow researchers to determine which microbial taxa are affected over time by perturbations such as onset of disease or lifestyle choices, and to predict the effects of these perturbations over time, including changes in composition or stability of commensal bacteria.

Author(s)

Maintainer: Aubrey Odom <aodom@bu.edu> (ORCID)

Other contributors:

- Yilong Zhang <elong0527@gmail.com> (Author of NMIT functions) [contributor]
- Jared Pincus <pincus@bu.edu> (ORCID) (Naming consultant of package) [consultant]
- Jordan Pincus <pincus@bu.edu> (Artist of LegATo icon) [artist]

See Also

Useful links:

- https://wejlab.github.io/LegATo-docs/
- Report bugs at https://github.com/wejlab/LegATo/issues

`clean_MAE` *Clean up taxon names in a MultiAssayExperiment*

Description

This functional is an optional method for fixing up taxon names in a `MultiAssayExperiment` to be run before `filter_MAE`. Specifically, it removes brackets from species names, replaces species labeled as "others" with "sp." and finally replaces underscores with spaces.

Usage

```
clean_MAE(dat)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.

Value

An animalcules-formatted `MultiAssayExperiment` object with cleaned-up taxonomy nomenclature.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
clean_MAE(in_dat)
```

`create_formatted_MAE` *Create a formatted MultiAssayExperiment compatible with LegATo*

Description

This function takes either a counts_dat, tax_dat, and metadata_dat input OR a TreeSummarized-Experiment input and creates a specifically-formatted MAE object that is compatible for use with LegATo and animalcules. Checks are performed on inputs to ensure that they can be integrated properly.

Usage

```
create_formatted_MAE(
  counts_dat = NULL,
  tax_dat = NULL,
  metadata_dat = NULL,
  tree_SE = NULL
)
```

Arguments

`counts_dat`	A matrix, data.table, or data.frame consisting of microbial raw counts data. The `colnames` should be sample names and the `rownames` should be in the same order as the `tax_dat` entries. Not required if `tree_SE` is passed in.
`tax_dat`	A matrix, data.table, or data.frame of hierarchical taxonomic data. Should have columns such as "family", "genus", "species" with each row uniquely delineating a different taxon. The rows should be in the same order as the rows of `counts_dat`. Not required if `tree_SE` is passed in.
`metadata_dat`	A metadata table with `rownames` equivalent to the samples that are the `colnames` of the `counts_dat`. Not required if `tree_SE` is passed in.
`tree_SE`	A TreeSummarizedExperiment object with counts, taxonomy, and metadata.

Value

A `MultiAssayExperiment` object.

Examples

```
nsample <- ntaxa <- 3
counts_dat <- data.frame(
    "X123" = runif(ntaxa, 0, 500),
    "X456" = runif(ntaxa, 0, 500),
    "X789" = runif(ntaxa, 0, 500)
)
tax_dat <- data.frame(
    "class" = c("rand1", "rand2", "rand3"),
    "species" = c("rand4", "rand5", "rand6")
) |>
    as.data.frame()
# Set rownames as lowest unique taxonomic level
rownames(tax_dat) <- tax_dat$species
```

```
rownames(counts_dat) <- tax_dat$species
metadata <- data.frame(
    Sample = c("X123", "X456", "X789"),
    Group = c("A", "B", "A"),
    Var = rnorm(nsample)
)
rownames(metadata) <- metadata$Sample
out_MAE <- create_formatted_MAE(counts_dat, tax_dat, metadata)

# TreeSummarizedExperiment
tse <- TreeSummarizedExperiment::TreeSummarizedExperiment(
    assays = list(counts = counts_dat),
    colData = metadata,
    rowData = tax_dat
)
out_MAE_2 <- create_formatted_MAE(tree_SE = tse)
out_MAE_2
```

`distinctColors` *Generate a distinct palette for coloring different clusters.*

Description

Create a distinct palette for coloring different heatmap clusters. The function returns colors for input into `ComplexHeatmap:Heatmap()`. The "grDevices" package is required to use this function.

Usage

```
distinctColors(
  n,
 hues = c("red", "cyan", "orange", "blue", "yellow", "purple", "green", "magenta"),
  saturation.range = c(0.7, 1),
  value.range = c(0.7, 1)
)
```

Arguments

`n`	an integer describing the number of colors to generate. Required.
`hues`	a vector of character strings indicating the R colors available from the `colors()` function. These will be used as the base colors for the clustering scheme. Different saturations and values (i.e. darkness) will be generated for each hue. Default is `c("red", "cyan", "orange","blue", "yellow", "purple", "green", "magenta")`
`saturation.range`	a numeric vector of length 2 with values between 0 and 1 giving the range of saturation. The default is `c(0.25, 1)`.
`value.range`	a numeric vector of length 2 with values between 0 and 1 giving the range of values. The default is `c(0.5, 1)`.

Value

A vector of distinct colors that have been converted to HEX from HSV.

Examples

```
distinctColors(10)
```

`filter_animalcules_MAE` *Filter a MultiAssayExperiment to a top percentage of taxa and label the rest as "Other"*

Description

This function takes an animalcules-formatted `MultiAssayExperiment` (MAE) object and identifies all taxa at the "genus" level that represent <`filter_prop` average relative abundance across all samples in the MAE. After identification at the genus level, taxa across the genus and species levels are then consolidated into the category "Other".

Usage

```
filter_animalcules_MAE(dat, filter_prop = 0.001)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.
`filter_prop`	A double strictly between 0 and 1, representing the proportion of relative abundance at which to filter. Default is 0.001.

Value

An animalcules-formatted `MultiAssayExperiment` object with appropriate filtration.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
filter_animalcules_MAE(in_dat, 0.01)
```

`filter_MAE` *Filter a MultiAssayExperiment object to keep a top percentage of taxa*

Description

This function takes an animalcules-formatted `MultiAssayExperiment` (MAE) object and identifies all taxa at the OTU level of choice that exhibit a relative abundance greater than or equal to a relative abundance percent threshold, `relabu_threshold`, in at least `occur_pct_cutoff`% of the total samples. After filtration, taxa across the specified OTU level and all downstream levels are then consolidated into the category "Other".

Usage

```
filter_MAE(
  dat,
  relabu_threshold = 3,
  occur_pct_cutoff = 5,
  taxon_level = "genus"
)
```

Arguments

dat	A `MultiAssayExperiment` object specially formatted as an animalcules output.
relabu_threshold	A double(percentage) between 0 and 100, representing the relative abundance criterion that all OTUs should meet to be retained. The smaller the threshold, the fewer the OTUs will be retained. Default is 3%.
occur_pct_cutoff	A double (percentage) between 0 and 100 representing the percent cutoff for how many OTUs must meet the `relabu_threshold` across the samples to be retained. It is wise to keep the number of samples in mind when setting this parameter. Default is 5%.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in `MultiAssayExperiment::rowData(dat)`.

Value

An animalcules-formatted `MultiAssayExperiment` object with major OTUs retained.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |>
  readRDS()
filter_MAE(in_dat, relabu_threshold = 3, occur_pct_cutoff = 5,
           taxon_level = "genus")
```

---

get_long_data *Create a long data.frame from a MultiAssayExperiment counts object*

Description

This function takes a `MultiAssayExperiment` object and a specified taxon level of interest and creates a long `data.frame` that can be used more easily for plotting counts data.

Usage

```
get_long_data(dat, taxon_level, log = FALSE, counts_to_CPM = FALSE)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).
log	logical. Indicate whether an assay returned should be the log of whichever assay is specified in "output_name". If counts_to_CPM = TRUE as well, then a log CPM assay will also be created. Default is FALSE.
counts_to_CPM	logical. This argument only applies if the input_type is a counts assay. If TRUE, then the output assays will include a normalized CPM assay. If log = TRUE as well, then a log CPM assay will also be created. Default is TRUE.

Value

A data.frame consisting of the counts data, taxa, and metadata.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
out <- get_long_data(in_dat, "genus", log = TRUE, counts_to_CPM = TRUE)
head(out)
```

---

get_stacked_data *Create a long data.frame with grouped abundances from a MultiAssayExperiment counts object*

Description

This function takes a MultiAssayExperiment object and a specified taxon level of interest and creates a long data.frame that can be used more easily for plotting counts data in a stacked bar plot or a stacked area chart. The function groups taxa and computes relative abundance within taxa strata.

Usage

```
get_stacked_data(dat, taxon_level = "genus", covariate_1, covariate_time)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).
covariate_1	Character string, the name of the covariate in ‘dat‘ by which to color and group samples. Default is NULL.
covariate_time	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.

Value

A data.frame consisting of the counts data, taxa, and metadata.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
get_stacked_data(in_dat, "genus", covariate_1 = "Sex", covariate_time = "Month")
```

`get_summary_table` *Create a table summarizing reads aggregated by grouping variables on a unit*

Description

This function takes a `MultiAssayExperiment` of microbial read counts and aggregates them by one or more grouping vars within a unit.

Usage

```
get_summary_table(dat, group_vars = NULL)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.
`group_vars`	A character string or character vector of covariates found in `colData(dat)` to use in grouping counts. The variables should be listed in order of desired grouping. Default is `NULL`, which does not rely on a grouping variable and instead produces statistics for the entirety of the data.

Value

A `data.frame` of the grouping columns, mean_reads, sd_reads, min_reads, max_reads and num_total.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
out <- get_summary_table(in_dat, c("Group", "Subject"))
head(out)
```

`get_top_taxa` *Obtain a data.frame of ordered taxa abundances at a given level*

Description

This function takes a `MultiAssayExperiment` object and returns a data.frame of the present taxa at a user-supplied taxonomy level, and outputs the average abundances of the taxa.

Usage

```
get_top_taxa(dat, taxon_level = "genus")
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).

Value

A data.frame

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
out <- get_top_taxa(in_dat, "genus")
out
```

NMIT *Nonparametric Microbial Interdependence Test (NMIT)*

Description

An R-based implementation of the NMIT, a multivariate distance-based test for group comparisons of microbial temporal interdependence. The NMIT test provides a comprehensive way to evaluate the association between key phenotypic variables and microbial interdependence. This function is recommended for use after a filtering step using filter_MAE. Note, the "ComplexHeatmap" package is required to use the plotting features of the function. The function requires the "vegan" package.

Usage

```
NMIT(
  dat,
  unit_var,
  fixed_cov,
  covariate_time,
  method = "kendall",
  dist_type = "F",
  heatmap = TRUE,
  classify = FALSE,
  fill_na = 0,
  ...
)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
unit_var	a numeric vector of subject.
fixed_cov	A character vector of the names of covariates of interest found in dat.
covariate_time	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.

method	an option of the correlation method ("pearson", "kendall", "spearman"). The default method is "kendall".
dist_type	A character string specifying the type of matrix norm to be computed. The default is "F". * "M" or "m" specifies the maximum modulus of all the elements in x; * "O", "o" or "1" specifies the one norm, (maximum absolute column sum); * "I" or "i" specifies the infinity norm (maximum absolute row sum); * "F" or "f" specifies the Frobenius norm (the Euclidean norm of x treated as if it were a vector)
heatmap	A logical value indicating whether to draw heatmap. The default is TRUE.
classify	A logical value indicating whether to draw a classifier tree. The default is FALSE.
fill_na	A number between 0 and 1 to fill NA values. The default value is 0.
...	Additional arguments to be passed to ComplexHeatmap::Heatmap().

Value

This function returns an analysis of variance (ANOVA) table showing sources of variation, degrees of freedom, sequential sums of squares, mean squares, F statistics, partial R-squared and P values, based on 999 permutations.

Author(s)

Yilong Zhang, Huilin Li, Aubrey Odom

Examples

```
dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
NMIT(dat, unit_var = "Subject", fixed_cov = "Group", covariate_time = "Month")
```

---

parse_MAE_SE *Parse a MultiExperimentAssay object and extract the elements as data.frames*

Description

This function takes an animalcules-formatted MultiAssayExperiment object and parses it to extract a named assay alongside the taxonomy and metadata.

Usage

```
parse_MAE_SE(dat, which_assay = NULL, type = "MAE")
```

Arguments

dat	Either a MultiAssayExperiment or a SummarizedExperiment object.
which_assay	Character string. This refers to the assay to be extracted from the MultiAssayExperiment object if type = "MAE". Does not need to be specified if type = "SE". Default is NULL.
type	One of "MAE" denoting a MultiAssayExperiment or "SE" denoting a SummarizedExperiment.

Value

Returns a list of 3 named data.frame elements, ‘counts‘, ‘sam‘, and ‘tax‘ denoting the counts data, sample metadata table, and taxonomy table, respectively.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
out <- parse_MAE_SE(in_dat)
head(out$tax)
head(out$sam)
head(out$counts)

out2 <- parse_MAE_SE(in_dat[["MicrobeGenetics"]],
                     which_assay = "MGX", type = "SE")
```

---

plot_alluvial *Plot an alluvial diagram of microbial relative abundance*

Description

This function takes a `MultiAssayExperiment` object and returns a alluvial diagram of microbe relative abundances. The function takes a single covariate as an optional variable by which to create a grid of multiple plots. Note, the ggalluvial package is required to use this function.

Usage

```
plot_alluvial(
  dat,
  taxon_level,
  covariate_1 = NULL,
  covariate_time,
  palette_input = NULL,
  title = paste("Relative abundance at", taxon_level, "level"),
  subtitle = NULL
)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.
`taxon_level`	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in `MultiAssayExperiment::rowData(dat)`.
`covariate_1`	Character string giving the name of a column in `MultiAssayExperiment::colData(dat)` on which to create multiple plots. The default is `NULL`.
`covariate_time`	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.
`palette_input`	A character vector of colors that is at minimum the same length of the number of taxa (specified with `taxon_level`). The default is `NULL` and relies on `ggplot2`'s default scheme.
`title`	A character string providing the plot title.
`subtitle`	A character string providing the plot subtitle. The default is `NULL`.

Details

If further manipulation of specific parameters is desired, users can add ggplot2 function calls to the output of the function.

Value

A ggplot2 plot.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
plot_alluvial(in_dat, taxon_level = "family", covariate_1 = "Group", covariate_time = "Month",
              palette_input = rainbow(25))
```

---

plot_heatmap *Plot a ComplexHeatmap.*

Description

This function takes an arbitrary dataset as an input and returns a ComplexHeatmap plot of samples based on similarity of microbial abundances. The function takes arguments listed here as well as any others to be passed on to ComplexHeatmap::Heatmap(). Note, the "circlize" and "ComplexHeatmap" packages are required to use this function.

Usage

```
plot_heatmap(
  inputData,
  annotationData = NULL,
  plot_title = NULL,
  name = "Input data",
  plottingColNames,
  annotationColNames = NULL,
  colList = list(),
  scale = FALSE,
  showColumnNames = TRUE,
  showRowNames = TRUE,
  colorSets = c("Set1", "Set2", "Set3", "Pastel1", "Pastel2", "Accent", "Dark2",
    "Paired"),
  choose_color = c("blue", "gray95", "red"),
  split_heatmap = "none",
  annotationplotting = NULL,
  column_order = NULL,
  ...
)
```

Arguments

inputData	an input data object. It should either be of the class SummarizedExperiment and contain the data and annotation data as columns in the colData, or alternatively be of the classes data.frame or matrix and contain only the plotting data. Required.
annotationData	a data.frame or matrix of annotation data, with one column. Only required if inputData is a data.frame or matrix of plotting data. The row names must equal those of the inputData column names. Default is NULL.
plot_title	a character string with the plot title of the heatmap. The default is NULL.
name	a character string with the name of the data to be displayed. Default is "Input data".
plottingColNames	a vector of the column names in colData that contain the plotting data. Only required if inputData is a SummarizedExperiment object.
annotationColNames	a vector of the column names in colData that contain the annotation data. Only required if inputData is a SummarizedExperiment. Default is NULL.
colList	a named list of named vectors specifying custom color information to pass to ComplexHeatmap::Heatmap(). The list should have as many elements as there are annotation columns, and each element name should correspond exactly with the name of each annotation column. The colors in the vector elements should be named according to the levels of the factor in that column's annotation data if the annotation is discrete, or it should be produced with circlize::colorRamp2 if the annotation is continuous. By default, ColorBrewer color sets will be used. See the the parameter colorSets for additional details.
scale	logical. Setting scale = TRUE scales the plotting data. The default is FALSE.
showColumnNames	logical. Setting showColumnNames = TRUE will show the column names (i.e. sample names) on the heatmap. The default is TRUE.
showRowNames	logical. Setting showColumnNames = TRUE will show the row names (i.e. plotting names) on the heatmap. The default is TRUE.
colorSets	a vector of names listing the color sets in the order that they should be used in creating the heatmap. By default, this function will use the color sets in the order listed in Usage for annotation information. You may replace the default with the same collection of sets in order that you want to use them, or provide custom color sets with the colList parameter.
choose_color	a vector of color names to be interpolated for the heatmap gradient, or a colorRamp function produced by circlize::colorRamp2. The default is c("blue", "gray95", "red").
split_heatmap	a character string either giving the column title of annotationplotting containing annotation data for which to split the heatmap rows, or "none" if no split is desired.
annotationplotting	a data.frame or matrix with information to be used in splitting the heatmap. The first column should plotting names. The column of annotation information should be specified in split_heatmap. Other columns will be ignored. The default is sigAnnotData.
column_order	a vector of character strings indicating the order in which to manually arrange the heatmap columns. Default is NULL, such that column order is automatically determined via clustering.

... Additional arguments to be passed to `ComplexHeatmap::Heatmap()`.

Details

If both `annotationData = NULL` and `annotationColNames = NULL`, no annotation bar will be drawn on the heatmap.

Code was adapted from the `TBSignatureprofiler` R package.

Value

A ComplexHeatmap plot.

Author(s)

David Jenkins, Aubrey Odom

Examples

```
library(SummarizedExperiment)
# Example with a Summarized Experiment data object
dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
input_SE <- dat[["MicrobeGenetics"]]

## Creating a continuous color ramp annot col list
Hairrange <- range(colData(input_SE)[, "HairLength"])
color2 <- circlize::colorRamp2(c(Hairrange[1], Hairrange[2]), c("blue", "red"))
color.list <- list("HairLength" = color2)

## Create plot
plot_heatmap(
  inputData = input_SE,
  name = "Microbe abundances",
  plot_title = "Example Heatmap",
  plottingColNames = colnames(input_SE),
  annotationColNames = "HairLength",
  colList = color.list,
  scale = TRUE,
  showColumnNames = TRUE,
  showRowNames = FALSE,
  colorSets =
    c("Set1", "Set2", "Set3", "Pastel1", "Pastel2", "Accent", "Dark2",
      "Paired"),
  choose_color = c("blue", "gray95", "red"),
  split_heatmap = "none",
  column_order = NULL
)

# Artificial data example - matrix input
mat_testdata <- rbind(matrix(c(rnorm(80), rnorm(80) + 5), 16, 10,
                             dimnames = list(paste0("Taxon", seq_len(16)),
                                             paste0("sample", seq_len(10)))),
                      matrix(rnorm(1000), 100, 10,
                             dimnames = list(paste0("Taxon0", seq_len(100)),
                                             paste0("sample", seq_len(10)))))
cov_mat <- data.frame(sample = c(rep("down", 5), rep("up", 5))) |>
  magrittr::set_rownames(paste0("sample", seq_len(10)))
```

```
# Example using custom colors for the annotation information
color2 <- stats::setNames(c("purple", "black"), c("down", "up"))
color.list <- list("sample" = color2)

plot_heatmap(
  inputData = mat_testdata,
  annotationData = cov_mat,
  name = "Data",
  plot_title = "Example",
  plottingColNames = NULL,
  annotationColNames = NULL,
  colList = color.list,
  scale = FALSE,
  showColumnNames = TRUE,
  showRowNames = FALSE,
  colorSets = c("Set1", "Set2", "Set3", "Pastel1", "Pastel2", "Accent", "Dark2",
                "Paired"),
  choose_color = c("blue", "gray95", "red"),
  split_heatmap = "none",
  column_order = NULL
)
```

`plot_spaghetti` *Plot a spaghetti volatility plot of microbial abundance for a given taxon*

Description

This function takes a `MultiAssayExperiment` object and returns a spaghetti plot of microbial abundance delineated by a unit, such as a subject.

Usage

```
plot_spaghetti(
  dat,
  covariate_time,
  covariate_1 = NULL,
  unit_var,
  taxon_level,
  which_taxon,
  palette_input = NULL,
  title = "Spaghetti Plot",
  subtitle = NULL
)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.
`covariate_time`	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.

covariate_1	Character string, the name of the covariate in 'dat' by which to color and group samples. Default is NULL.
unit_var	Character string, the name of the column delineating the unit on which the microbial abundances are changing over time. This is likely something akin to a subject that repeated measurements are made on.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).
which_taxon	Character string, the name of the taxon to plot at the specified taxon level.
palette_input	A character vector of colors that is at minimum the same length of the number of taxa (specified with taxon_level). The default is NULL and relies on ggplot2's default scheme.
title	A character string providing the plot title.
subtitle	A character string providing the plot subtitle. The default is NULL.

Details

If further manipulation of specific parameters is desired, users can add ggplot2 function calls to the output of the function.

Value

A ggplot2 plot.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
all_taxa <- get_top_taxa(in_dat, "phylum")
plot_spaghetti(in_dat, taxon_level = "phylum", covariate_1 = "Group", covariate_time = "Month",
              unit_var = "Subject", which_taxon = all_taxa$taxon[1],
              palette_input = rainbow(25))
```

---

plot_stacked_area *Plot a stacked area chart of microbial relative abundance*

Description

This function takes a MultiAssayExperiment object and returns a stacked area chart of microbe relative abundances. The function takes a single covariate as an optional variable by which to create a grid of multiple plots.

Usage

```
plot_stacked_area(
  dat,
  taxon_level,
  covariate_1 = NULL,
  covariate_time,
  palette_input = NULL,
  title = paste("Relative abundance at", taxon_level, "level"),
  subtitle = NULL
)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).
covariate_1	Character string giving the name of a column in MultiAssayExperiment::colData(dat) on which to create multiple plots. The default is NULL.
covariate_time	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.
palette_input	A character vector of colors that is at minimum the same length of the number of taxa (specified with taxon_level). The default is NULL and relies on ggplot2's default scheme.
title	A character string providing the plot title.
subtitle	A character string providing the plot subtitle. The default is NULL.

Details

If further manipulation of specific parameters is desired, users can add ggplot2 function calls to the output of the function.

Value

A ggplot2 plot.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
plot_stacked_area(in_dat, taxon_level = "phylum", covariate_1 = "Group",
                  covariate_time = "Month",
                  palette_input = rainbow(25))
```

---

plot_stacked_bar *Plot a stacked bar chart of microbial relative abundance*

Description

This function takes a MultiAssayExperiment object and returns a stacked bar plot of microbe relative abundances. The function takes a single covariate as an optional variable by which to create multiple gridded plots.

Usage

```
plot_stacked_bar(
  dat,
  taxon_level,
  covariate_1 = NULL,
  covariate_time,
  palette_input = NULL,
  title = paste("Relative abundance at", taxon_level, "level"),
  subtitle = NULL
)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string indicating the level of taxonomy to aggregate the counts data. Must be the name of a column in MultiAssayExperiment::rowData(dat).
covariate_1	Character string giving the name of a column in MultiAssayExperiment::colData(dat) on which to create multiple plots. The default is NULL.
covariate_time	Character string giving the name of the discrete time-based covariate in the metadata to group abundances by.
palette_input	A character vector of colors that is at minimum the same length of the number of taxa (specified with taxon_level). The default is NULL and relies on ggplot2's default scheme.
title	A character string providing the plot title.
subtitle	A character string providing the plot subtitle. The default is NULL.

Details

If further manipulation of specific parameters is desired, users can add ggplot2 function calls to the output of the function.

Value

A ggplot2 plot.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
plot_stacked_bar(in_dat, taxon_level = "family", covariate_1 = "Group",
                 covariate_time = "Month",
                 palette_input = rainbow(25))
```

---

run_gee_model *Compute Generalized Estimating Equations (GEEs) on longitudinal microbiome data*

Description

This function takes an animalcules-formatted MultiAssayExperiment and runs an independent GEE model for each taxon. The model predicts taxon log CPM abundance as a product of fixed-effects covariates conditional on a grouping ID variable, usually the unit on which repeated measurements were taken. This modeling approach works best with small datasets that multiple samples across many (>40) clusters/units. Note, the "broom", "ggeffects", "broom.mixed", "geepack", "emmeans" packages are required to use this function; all can be installed via CRAN.

Usage

```
run_gee_model(
  dat,
  taxon_level = "genus",
  unit_var,
  fixed_cov,
  corstr = "ar1",
  p_adj_method = "fdr",
  plot_out = FALSE,
  plotsave_loc = ".",
  plot_terms = NULL,
  ...
)
```

Arguments

dat	A MultiAssayExperiment object specially formatted as an animalcules output.
taxon_level	Character string, default is "genus".
unit_var	Character string giving the variable containing the identifiers for the unit on which multiple measurements were conducted, e.g. subjects. Default is NULL; must be supplied if paired = FALSE.
fixed_cov	A character vector naming covariates to be tested.
corstr	A character string specifying the correlation structure. The following are permitted: '"independence"', '"exchangeable"', '"ar1"', '"unstructured"'.
p_adj_method	A character string specifying the correction method. Can be abbreviated. See details. Default is "fdr".
plot_out	Logical indicating whether plots should be output alongside the model results. Default is FALSE.
plotsave_loc	A character string giving the folder path to save plot outputs. This defaults to the current working directory.
plot_terms	Character vector. Which terms should be examined in the plot output? Can overlap with the fixed_cov inputs.
...	Further arguments passed to ggsave for plot creation.

Details

P-values are adjusted for the model coefficients within each taxon. The following methods are permitted: c("holm", "hochberg", "hommel","bonferroni", "BH", "BY", "fdr", "none")

Value

A data.frame of modeling results.

Examples

```
in_dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |>
            readRDS()
out <- run_gee_model(in_dat, taxon_level = "genus", unit_var = "Subject",
                     fixed_cov = c("HairLength", "Age", "Group", "Sex"),
                     corstr = "ar1")
head(out)
```

`run_lmm_model` *Compute linear mixed-effects models (LMM) on longitudinal microbiome data*

Description

This function takes an animalcules-formatted `MultiAssayExperiment` and runs an independent LMM model for each taxon. The model predicts taxon log CPM abundance as a product of fixed-effects covariates with a random effect, usually the unit on which repeated measurements were taken. Note, the 'broom', 'lmerTest', and 'broom.mixed' packages are required to use this function; they can be downloaded from CRAN.

Usage

```
run_lmm_model(
  dat,
  taxon_level = "genus",
  unit_var,
  fixed_cov,
  p_adj_method = "fdr",
  plot_out = FALSE,
  plotsave_loc = ".",
  plot_terms = NULL,
  ...
)
```

Arguments

`dat`	A MultiAssayExperiment object specially formatted as an animalcules output.
`taxon_level`	Character string, default is `"genus"`.
`unit_var`	Character string giving the variable containing the identifiers for the unit on which multiple measurements were conducted, e.g. subjects. Default is `NULL`; must be supplied if `paired = FALSE`.
`fixed_cov`	A character vector naming covariates to be tested.
`p_adj_method`	A character string specifying the correction method. Can be abbreviated. See details. Default is `"fdr"`.
`plot_out`	Logical indicating whether plots should be output alongside the model results. Default is `FALSE`.
`plotsave_loc`	A character string giving the folder path to save plot outputs. This defaults to the current working directory.
`plot_terms`	Character vector. Which terms should be examined in the plot output? Can overlap with the `fixed_cov` inputs.
`...`	Further arguments passed to `ggsave` for plot creation.

Details

P-values are adjusted for the model coefficients within each taxon. The following methods are permitted: `c("holm", "hochberg", "hommel","bonferroni", "BH", "BY", "fdr", "none")`

Value

A `data.frame` of modeling results.

Examples

```
dat <- system.file("extdata/MAE.RDS", package = "LegATo") |>
  readRDS() |>
  filter_MAE()
out <- run_lmm_model(dat, taxon_level = "genus", unit_var = "Subject",
                     fixed_cov = c("HIVStatus", "timepoint"))
head(out)
```

`run_lm_model` *Compute linear models (LM) on microbiome data*

Description

This function takes an animalcules-formatted `MultiAssayExperiment` and runs an independent linear model for each taxon. The model predicts taxon log CPM abundance as a product of user-specified covariates. This model can be used for general microbiome analyses without repeated measures data. Note, the "broom", and "broom.mixed" packages are required to use the testing functionality of this package; both can be installed via CRAN.

Usage

```
run_lm_model(
  dat,
  taxon_level = "genus",
  fixed_cov,
  p_adj_method = "fdr",
  plot_out = FALSE,
  plotsave_loc = ".",
  plot_terms = NULL,
  ...
)
```

Arguments

`dat`	A MultiAssayExperiment object specially formatted as an animalcules output.
`taxon_level`	Character string, default is `"genus"`.
`fixed_cov`	A character vector naming covariates to be tested.
`p_adj_method`	A character string specifying the correction method. Can be abbreviated. See details. Default is `"fdr"`.
`plot_out`	Logical indicating whether plots should be output alongside the model results. Default is `FALSE`.
`plotsave_loc`	A character string giving the folder path to save plot outputs. This defaults to the current working directory.
`plot_terms`	Character vector. Which terms should be examined in the plot output? Can overlap with the `fixed_cov` inputs.
`...`	Further arguments passed to `ggsave` for plot creation.

Details

P-values are adjusted for the model coefficients within each taxon. The following methods are permitted: `c("holm", "hochberg", "hommel","bonferroni", "BH", "BY", "fdr", "none")`

Value

A `data.frame` of modeling results.

Examples

```
dat <- system.file("extdata/MAE.RDS", package = "LegATo") |>
  readRDS() |>
  filter_MAE()
out <- run_lm_model(dat, fixed_cov = c("timepoint", "HIVStatus"),
                    plot_out = FALSE)
head(out)
```

---

test_hotelling_t2 *Conduct a multivariate Hotelling's T-squared test*

Description

This function takes an animalcules-formatted `MultiAssayExperiment` object and runs a multivariate Hotelling's T-squared test. The test expects a comparison of two distinct groups, and compares the abundances of the top microbes at a given taxon level between the groups. This function allows both paired and unpaired tests. Both test the null hypothesis that the population mean vectors are equal, with the alternative being that they are unequal.

Usage

```
test_hotelling_t2(
  dat,
  test_index = NULL,
  taxon_level = "genus",
  num_taxa,
  grouping_var,
  paired = FALSE,
  pairing_var = NULL,
  unit_var = NULL,
  save_table_loc = "."
)
```

Arguments

`dat`	A MultiAssayExperiment object specially formatted as an animalcules output.
`test_index`	Any argument used for subsetting the input `dat`, can be a character, logical, integer, list or List vector. Default is `NULL`.
`taxon_level`	Character string, default is `"genus"`.

num_taxa	The number of most abundant taxa to test. If unpaired, this should be no larger than the total number of subjects in both groups - 2, or (n1 + n2 -2). If paired, this should be no larger than the total number of pairs - 1, or n - 1. Required.
grouping_var	Character string, the name of a DICHOTOMOUS grouping variable in the metadata of dat.
paired	Logical indicating whether a paired test should be conducted. Default is FALSE for an unpaired test.
pairing_var	Character string giving the variable containing pairing information. The variable should be in integer form. Must be supplied if paired = TRUE, otherwise the default is NULL.
unit_var	Character string giving the variable containing the identifiers for the unit on which multiple measurements were conducted, e.g. subjects. Default is NULL; must be supplied if paired = FALSE.
save_table_loc	A character string giving the folder path to save t.test results. Note that these are only conducted if the Hotelling's T-test value is <0.05. Defaults to the current working directory.

Details

The Hotelling's t-squared statistic (t2) is a generalization of Student's t-statistic that is used in multivariate hypothesis testing to test the means of different populations.

Note that any entries or pairs with missing values are excluded.

Referenced articles in the implementation of tests:

https://online.stat.psu.edu/stat505/lesson/7/7.1/7.1.14

https://online.stat.psu.edu/stat505/lesson/7/7.1/7.1.15

https://online.stat.psu.edu/stat505/lesson/7/7.1/7.1.4

https://online.stat.psu.edu/stat505/lesson/7/7.1/7.1.9

Value

A list of the elements "df1", "df2", "crit_F", "F_stat" and "pvalue" giving the results of the test.

Examples

```
dat <- system.file("extdata", "MAE.RDS", package = "LegATo") |>
readRDS()
dat_0.05 <- filter_MAE(dat, 0.001, 10, "species")
out1 <- test_hotelling_t2(dat = dat_0.05,
                  test_index = which(dat_0.05$MothChild == "Infant" &
                                       dat_0.05$timepoint == 0),
                  taxon_level = "genus",
                  # Total number of pairs - 1
                  num_taxa = 9,
                  paired = TRUE,
                  grouping_var = "HIVStatus",
                  pairing_var = "pairing")
out1

out <- test_hotelling_t2(dat = dat_0.05,
                  test_index = which(dat_0.05$MothChild == "Mother" &
                                       dat_0.05$timepoint == 6),
```

```
                    taxon_level = "genus",
                    # Max is Total number of subjects - 2
                    # Here we use a much smaller number
                    num_taxa = 6,
                    grouping_var = "HIVStatus",
                    unit_var = "Subject",
                    paired = FALSE)
out
```

`tscor` *Calculate within-subject OTU correlations*

Description

This function takes a `MultiAssayExperiment` and outputs an array of temporal intra-subject correlation matrices.

Usage

```
tscor(dat, unit_var, method = "kendall", fill_na = 0)
```

Arguments

`dat`	A `MultiAssayExperiment` object specially formatted as an animalcules output.
`unit_var`	a numeric vector of subject.
`method`	an option of the correlation method ("pearson", "kendall", "spearman"). The default method is "kendall".
`fill_na`	a number between 0 and 1 to fill the missing value. The default value is 0.

Value

An three-dimensional array of temporal correlation matrices for each subject.

Author(s)

Yilong Zhang, Huilin Li, Aubrey Odom

Examples

```
dat <- system.file("extdata/MAE_small.RDS", package = "LegATo") |> readRDS()
output <- tscor(dat, unit_var = "Subject", method = "spearman")
head(output)
```

Index