Package 'dar'

November 11, 2026

Title Differential Abundance Analysis by Consensus

Version 1.8.0

Date 2025-10-14

Description Differential abundance testing in microbiome data challenges both parametric and non-parametric statistical methods, due to its sparsity, high variability and compositional nature. Microbiome-specific statistical methods often assume classical distribution models or take into account compositional specifics. These produce results that range within the specificity vs sensitivity space in such a way that type I and type II error that are difficult to ascertain in real microbiome data when a single method is used. Recently, a consensus approach based on multiple differential abundance (DA) methods was recently suggested in order to increase robustness. With dar, you can use dplyr-like pipeable sequences of DA methods and then apply different consensus strategies. In this way we can obtain more reliable results in a fast, consistent and reproducible way.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(roclets = c(``collate", ``namespace", ``rd", ``roxytest::testthat_roclet", ``roxyglobals::global_roclet"), markdown = TRUE)

URL https://github.com/MicrobialGenomics-IrsicaixaOrg/dar, https://microbialgenomics-irsicaixaorg.github.io/dar/

BugReports https://github.com/MicrobialGenomics-IrsicaixaOrg/dar/issues

biocViews Software, Sequencing, Microbiome, Metagenomics, MultipleComparison, Normalization

Imports checkmate, cli, ComplexHeatmap, crayon, dplyr, generics, ggplot2, glue, gplots, heatmaply, magrittr, methods, mia, phyloseq, purrr, readr, rlang (>= 0.4.11), scales, stringr, tibble, tidyr, UpSetR

Suggests ALDEx2, ANCOMBC, apeglm, ashr, Biobase, corncob, covr, DESeq2, devtools, furrr, future, knitr, lefser, limma, maaslin3, microbiome, rmarkdown, roxygen2, roxyglobals,

roxytest, rstatix, SummarizedExperiment, TreeSummarizedExperiment, testthat (>= 3.0.0), GenomeInfoDb, withr

Config/testthat/edition 3

Depends R (>= 4.5.0)

LazyData false

Collate 'recipe-class.R' 'aldex2.R' 'ancom.R' 'bake.R' 'corncob.R' 'dar-package.R' 'data.R' 'deseq2.R' 'filter_by_abundance.R' 'filter_by_prevalence.R' 'filter_by_rarity.R' 'filter_by_variance.R' 'filter_taxa.R' 'globals.R' 'lefse.R' 'maaslin3.R' 'metagenomeseq.R' 'misc.R' 'phyloseq_qc.R' 'pkg_check.R' 'plot_methods.R' 'rarefaction.R' 'recipe-utils.R' 'steps_and_checks.R' 'subset_taxa.R' 'utilis-validation.R' 'utils-pipe.R' 'utils-tidy-eval.R' 'utils.R' 'wilcox.R'

VignetteBuilder knitr

Config/roxyglobals/filename globals.R

Config/roxyglobals/unique TRUE

Config/testthat/parallel true

RoxygenNote 7.3.3

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

git_branch RELEASE_3_23

git_last_commit df167e0

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Francesc Catala-Moll [aut, cre] (ORCID: <https://orcid.org/0000-0002-2354-8648>)

Maintainer Francesc Catala-Moll <fcatala@irsicaixa.es>

Contents

dar-package 4
abundance_plt 4
add_step 5
add_tax 6
add_var 7
bake 7
contains_rarefaction 9
cool 10
corr_heatmap 10
exclusion_plt 11
export_steps 12

find_intersections . . . 12
get_comparisons . . . 13
get_phy . . . 14
get_tax . . . 14
get_var . . . 15
import_steps . . . 15
intersection_df . . . 16
intersection_plt . . . 17
metaHIV_phy . . . 18
mutual_plt . . . 18
otu_table . . . 19
overlap_df . . . 20
pastry_df . . . 21
phyloseq_or_null-class . . . 21
phy_qc . . . 22
prep . . . 23
PrepRecipe-class . . . 24
prep_recipe . . . 24
rand_id . . . 25
rarefaction_help . . . 25
recipe . . . 26
recipes_pkg_check . . . 27
required_deps . . . 28
sample_data . . . 28
step . . . 29
steps_ids . . . 30
step_aldex . . . 30
step_ancom . . . 32
step_corncob . . . 35
step_deseq . . . 37
step_filter_by_abundance . . . 39
step_filter_by_prevalence . . . 40
step_filter_by_rarity . . . 42
step_filter_by_variance . . . 43
step_filter_taxa . . . 44
step_lefse . . . 45
step_maaslin . . . 47
step_metagenomeseq . . . 49
step_rarefaction . . . 49
step_subset_taxa . . . 50
step_wilcox . . . 51
tax_table . . . 53
test_prep_rec . . . 53
test_rec . . . 54
tidyeval . . . 54
to_tibble . . . 56
use_rarefy . . . 57
%>% . . . 58

Index **59**

`dar-package` *dar: Differential Abundance Analysis by Consensus*

Description

To learn more about dar, start with the vignettes: `browseVignettes(package = "dar")`

Author(s)

Maintainer: Francesc Catala-Moll <fcatala@irsicaixa.es> (ORCID)

See Also

Useful links:

- https://github.com/MicrobialGenomics-IrsicaixaOrg/dar
- https://microbialgenomics-irsicaixaorg.github.io/dar/
- Report bugs at https://github.com/MicrobialGenomics-IrsicaixaOrg/dar/issues

---

`abundance_plt` *Abundance boxplot*

Description

Abundance boxplot

Usage

```
abundance_plt(
  rec,
  taxa_ids = NULL,
  type = "boxplot",
  transform = "compositional",
  scale = 1,
  top_n = 20
)
```

Arguments

rec	A prepped Recipe object.
taxa_ids	Character vector with taxa_ids to plot. If taxa_ids is NULL the significant characteristics present in all of the executed methods will be plotted.
type	Character vector indicating the type of the result. Options: c("boxplot", "heatmap").
transform	Transformation to apply. The options include: 'compositional' (ie relative abundance), 'Z', 'log10', 'log10p', 'hellinger', 'identity', 'clr', 'alr', or any method from the vegan::decostand function. If the value is NULL, no normalization is applied and works with the raw counts.
scale	Scaling constant for the abundance values when transform = "scale".
top_n	Maximum number of taxa to represent. Default: 20.

Value

ggplot2 or HeatmapList

Examples

```
data(test_prep_rec)

## Running the function returns a boxplot,
abundance_plt(test_prep_rec)

## Giving the value "heatmap" to the type parameter, the resulting graph
## a heatmap.
# abundance_plt(test_prep_rec, type = "heatmap")

## By default, those taxa significant in all methods are plotted. If you want
## to graph some determined features, you can pass them as vector through the
## taxa_ids parameter.
# taxa_ids <- c("Otu_96", "Otu_78", "Otu_88", "Otu_35", "Otu_94", "Otu_34")
# abundance_plt(test_prep_rec, taxa_ids = taxa_ids)
# abundance_plt(test_prep_rec, taxa_ids = taxa_ids, type = "heatmap")

## abundance_plt function needs a PrepRecipe. If you pass a a non-prep
## Recipe the output is an error.
data(test_rec)
err <- testthat::expect_error(abundance_plt(test_rec))
err
```

---

add_step *Add a New Operation to the Current Recipe*

Description

add_step adds a step to the last location in the Recipe. add_check does the same for checks.

Usage

```
add_step(rec, object)
```

Arguments

rec	A Recipe() or PrepRecipe().
object	A step or check object.

Value

A updated Recipe() or PrepRecipe() with the new operation in the last slot.

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, "RiskGroup2", "Species")

# Internally, step_ functions use add_step to append themselves
rec <- step_maaslin(rec)
length(rec@steps) # Returns 1

# If we try to add the exact same step, it will inform us and skip it
rec <- step_maaslin(rec)
length(rec@steps) # Still returns 1
```

---

add_tax *Adds taxonomic level of interest in the Recipe.*

Description

Adds taxonomic level of interest in the Recipe.

Usage

```
add_tax(rec, tax_info)
```

Arguments

rec	A Recipe object (must not be prepped).
tax_info	A character string of taxonomic levels that will be used in any context.

Value

A Recipe object.

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy)
rec <- add_tax(rec, tax_info = "Species")
rec
```

---

add_var *Adds variable of interest to the Recipe*

Description

Adds variable of interest to the Recipe

Usage

```
add_var(rec, var_info)
```

Arguments

rec	A Recipe object (must not be prepped).
var_info	A character string of column names corresponding to variables that will be used in any context.

Value

A Recipe object.

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy)
rec <- add_var(rec, var_info = "RiskGroup2")
rec
```

---

bake *Define consensus strategies from a Recipe*

Description

For a prep Recipe adds a consensus strategy to use for result extraction.

Usage

```
bake(
  rec,
  count_cutoff = NULL,
  weights = NULL,
  exclude = NULL,
  id = rand_id("bake")
)
```

Arguments

rec	A PrepRecipe object. The step will be added to the sequence of operations for this Recipe.
count_cutoff	Indicates the minimum number of methods in which an OTU must be present (Default: NULL). If count_cutoff is NULL, it is set to length(steps_ids(rec, "da")) - length(exclude).
weights	Named numeric vector with the weight for each method.
exclude	Method ids to exclude.
id	A character string that is unique to this step to identify it.

Value

An object of class PrepRecipe

Examples

```
data(test_prep_rec)
rec <- test_prep_rec

## Default bake extracts common OTUs in all DA tested methods
## (In this case the Recipe contains 3 methods)
res <- bake(rec)
cool(res)

## bake() requires a PrepRecipe
data(test_rec)
err <- testthat::expect_error(bake(test_rec))
err

## We can use the parameter `count_cutoff` to select those OTUs
## shared by at least two methods
res <- bake(rec, count_cutoff = 2)
cool(res)

## Furthermore, we can exclude methods from the consensus strategy
## with the `exclude` parameter.
res <- bake(rec, exclude = steps_ids(rec, "da")[1])
cool(res)
```

```
## Finally, we can use the `weights` parameter to weigh each method.
weights <- c(2, 1)
names(weights) <- steps_ids(rec, "da")
res <- bake(rec, weights = weights)
cool(res)
```

`contains_rarefaction` *Checks if Recipe contains a rarefaction step*

Description

Checks if Recipe contains a rarefaction step

Usage

```
contains_rarefaction(rec)
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this recipe.

Value

boolean

Examples

```
data(GlobalPatterns, package = "phyloseq")
rec <-
  phyloseq::subset_samples(
    GlobalPatterns, SampleType %in% c("Soil", "Skin")
  ) |>
  recipe(var_info  = "SampleType", tax_info = "Genus") |>
  step_rarefaction()

contains_rarefaction(rec)
```

cool *Extract results from defined bake*

Description

Extract results from defined bake

Usage

```
cool(rec, bake = 1)
```

Arguments

rec	A PrepRecipe object.
bake	Name or index of the bake to extract.

Value

tbl_df

Examples

```
data(test_prep_rec)
rec <- bake(test_prep_rec)
cool(rec)
```

corr_heatmap *Plot otuput of the* overlap_df *function as a heatmap.*

Description

Plot otuput of the overlap_df function as a heatmap.

Usage

```
corr_heatmap(rec, steps = steps_ids(rec, "da"), font_size = 15, type = "all")
```

Arguments

rec	A prepped Recipe object.
steps	Character vector with step_ids to take in account.
font_size	Size of the axis font.
type	Indicates whether to use all taxa ("all") or only those that are differentially abundant in at least one method ("da"). Default as "all".

Value

heatmap

Examples

```
data(test_prep_rec)

## Running the function returns a UpSet plot ordered by frequency.
corr_heatmap(test_prep_rec)

## corr_heatmap function needs a PrepRecipe. If you pass a a non-prep
## Recipe the output is an error.
data(test_rec)
err <- testthat::expect_error(corr_heatmap(test_rec))
err
```

`exclusion_plt` *Plot the number of shared DA OTUs between methods.*

Description

Plot the number of shared DA OTUs between methods.

Usage

```
exclusion_plt(rec, steps = steps_ids(rec, "da"))
```

Arguments

`rec`	A prepped `Recipe` object.
`steps`	Character vector with step_ids to take in account.

Value

ggplot2-class object

Examples

```
data(test_prep_rec)

## Running the function returns a barplot plot,
exclusion_plt(test_prep_rec)

## If you want to exclude a method for the plot, you can remove it with the
## step parameter. In the following example we eliminate from the graph the
## results of maaslin
exclusion_plt(test_prep_rec, steps = steps_ids(test_prep_rec, "da")[-1])

## exclusion_plt function needs a PrepRecipe. If you pass a a non-prep
```

```
## Recipe the output is an error.
data(test_rec)
err <- testthat::expect_error(exclusion_plt(test_rec))
err
```

`export_steps` *Export step parameters as json.*

Description

Export step parameters as json.

Usage

```
export_steps(rec, file_name)
```

Arguments

`rec`	A Recipe object.
`file_name`	The path and file name of the optout file.

Value

invisible

`find_intersections` *Finds common OTU between method results*

Description

Finds common OTU between method results

Usage

```
find_intersections(rec, steps = steps_ids(rec, "da"))
```

Arguments

`rec`	A Recipe object.
`steps`	character vector with step ids to take in account

Value

tibble

Examples

```
data(test_prep_rec)

## From a PrepRecipe we can extract a tibble with all intersections
intersections <- find_intersections(test_prep_rec)
intersections

## Additionally, we can exclude some methods form the table
intersections <- find_intersections(
  test_prep_rec,
  steps = steps_ids(test_prep_rec, "da")[-1]
)

intersections
```

`get_comparisons` *Generate all unique contrasts between levels of a categorical variable.*

Description

Generate all unique contrasts between levels of a categorical variable.

Usage

```
get_comparisons(var, phy, as_list = TRUE, n_cut = 1)
```

Arguments

`var`	categorical variable
`phy`	phyloseq object
`as_list`	boolean indicating if output must be returned as a list.
`n_cut`	minimum of observations by level.

Value

tibble or list

Examples

```
data(test_rec)
dar:::get_comparisons("RiskGroup2", get_phy(test_rec))
```

get_phy *Returns phyloseq from Recipe-class object*

Description

Returns phyloseq from Recipe-class object

Usage

```
get_phy(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

Phyloseq class object

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
get_phy(rec)
```

get_tax *Returns tax_info from Recipe-class object*

Description

Returns tax_info from Recipe-class object

Usage

```
get_tax(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

Tibble containing tax_info.

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
get_tax(rec)
```

get_var *Returns var_info from Recipe-class object*

Description

Returns var_info from Recipe-class object

Usage

```
get_var(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

Tibble containing var_info.

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
get_var(rec)
```

import_steps *Import steps from json file*

Description

Import steps from json file

Usage

```
import_steps(
  rec,
  file,
  parallel = TRUE,
  workers = future::availableCores(constraints = "connections-16")
)
```

Arguments

rec	A Recipe object.
file	Path to the input file.
parallel	if FALSE, no palatalization. if TRUE, parallel execution using future and furrr packages.
workers	Number of workers for palatalization.

Value

recipe-class object

Examples

```
data(metaHIV_phy)

## Initialize the Recipe with a phyloseq object
rec <- recipe(metaHIV_phy, "RiskGroup2", "Species")
rec

## Import steps
json_file <- system.file("extdata", "test.json", package = "dar")
rec <- import_steps(rec, json_file)
rec

## If the json file contains 'bake', the Recipe is automatically prepared.
json_file <- system.file("extdata", "test_bake.json", package = "dar")
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  import_steps(json_file)

rec
cool(rec)
```

---

intersection_df *Returns data.frame with OTU intersection between methods*

Description

Returns data.frame with OTU intersection between methods

Usage

```
intersection_df(rec, steps = steps_ids(rec, "da"), tidy = FALSE)
```

Arguments

rec	A PrepRecipe object.
steps	character vector with step_ids to take in account.
tidy	Boolean indicating if result must be in tidy format.

Value

data.frame class object

Examples

```
data(test_prep_rec)
df <- intersection_df(test_prep_rec)
head(df)
```

`intersection_plt` *Plot results using UpSet plot*

Description

Plot results using UpSet plot

Usage

```
intersection_plt(
  rec,
  steps = steps_ids(rec, "da"),
  ordered_by = c("freq", "degree"),
  font_size = 2
)
```

Arguments

`rec`	A prepped `Recipe` object.
`steps`	Character vector with step_ids to take in account.
`ordered_by`	How the intersections in the matrix should be ordered by. Options include frequency (entered as "freq"), degree, or both in any order.
`font_size`	Size of the font.

Value

UpSet plot

Examples

```
data(test_prep_rec)

## Running the function returns a UpSet plot ordered by frequency.
intersection_plt(test_prep_rec)

## Alternatively, you can order the plot by degree
intersection_plt(test_prep_rec, ordered_by = "degree")

## intersection_plt function needs a PrepRecipe. If you pass a a non-prep
## Recipe the output is an error.
data(test_rec)
err <- testthat::expect_error(intersection_plt(test_rec))
err
```

metaHIV_phy *Phyloseq object from metaHIV project*

Description

A Phyloseq object containing abundance counts and sample_data for metaHIV project. Count reads were annotated with Metaphlan3.

Usage

```
data("metaHIV_phy")
```

Format

A phyloseq object with 451 taxas, 30 samples, 3 sample variables and 7 taxonomic ranks.

Source

s3://fcatala-09142020-eu-west-1/cloud_test/SpeciesQuantification/Kraken2

mutual_plt *Mutual finding plot*

Description

Plots number of differentially abundant features mutually found by defined number of methods, colored by the differential abundance direction and separated by comparison.

Usage

```
mutual_plt(
  rec,
  count_cutoff = NULL,
  comparisons = NULL,
  steps = steps_ids(rec, type = "da"),
  top_n = 20
)
```

Arguments

rec	A prepped Recipe object.
count_cutoff	Indicates the minimum number of methods in which an OTU must be present (Default: NULL). If count_cutoff is NULL count_cutoff is equal to length(steps_ids(rec, "da")) * 2 / 3.

comparisons	By default, this function plots all comparisons. However, if the user indicates the comparison or comparisons of interest, only the selected ones will be plotted.
steps	Character vector with step_ids to take in account. Default all "da" methods.
top_n	Maximum number of taxa to represent. Default: 20.

Value

ggplot2

Examples

```
data(test_prep_rec)

## Running the function returns a tile plot,
mutual_plt(test_prep_rec)

## The count_cutoff indicates the minimum number of methods in which an OTU
## must be present. By default the value is equal to
## length(steps_ids(rec, "da")) * 2 / 3 but it is customizable.
mutual_plt(
  test_prep_rec,
  count_cutoff = length(steps_ids(test_prep_rec, "da"))
)

## A single comparisons can be plotted through the comparison parameter.
mutual_plt(test_prep_rec, comparisons = c("hts_msm"))

## If you want to exclude a method for the plot, you can remove it with the
## step parameter. In the following example we eliminate from the graph the
## results of maaslin.
mutual_plt(test_prep_rec, steps = steps_ids(test_prep_rec, "da")[-1])

## mutual_plt function needs a PrepRecipe. If you pass a a non-PrepRecipe
## the output is an error.
data(test_rec)
err <- testthat::expect_error(mutual_plt(test_rec))
err
```

---

otu_table *Extracts otu_table from phyloseq inside a Recipe*

Description

Extracts otu_table from phyloseq inside a Recipe

Usage

```
otu_table(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

A tibble

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
otu_table(rec)
```

---

overlap_df *Overlap of significant OTUs between tested methods.*

Description

Overlap of significant OTUs between tested methods.

Usage

```
overlap_df(rec, steps = steps_ids(rec, "da"), type = "all")
```

Arguments

rec	A PrepRecipe object.
steps	Character vector with step_ids to take in account.
type	Indicates whether to use all taxa ("all") or only those that are differentially abundant in at least one method ("da"). Default as "all".

Value

df

Examples

```
data(test_prep_rec)
df <- overlap_df(test_prep_rec, steps_ids(test_prep_rec, "da"))
head(df)
```

pastry_df *Pastery data for step id generation*

Description

Tibble contain

Format

A tbl_df object with 228 unique pasteries.

Value

tibble with pastry names

Source

https://raw.githubusercontent.com/prasertcbs/basic-dataset/master/pastry.csv

phyloseq_or_null-class *Recipe-class object*

Description

A Recipe is a description of the steps to be applied to a data set in order to prepare it for data analysis.

Usage

```
## S4 method for signature 'PrepRecipe'
show(object)
```

Arguments

object	A Recipe object.

Value

Recipe-class object

Slots

`phyloseq` Phyloseq-class object.

`var_info` A tibble that contains the current set of terms in the data set. This initially defaults to the same data contained in `var_info`.

`tax_info` A tibble that contains the current set of taxonomic levels that will be used in the analysis.

`steps` List of step-class objects that will be used by DA.

---

phy_qc *Phyloseq Quality Control Metrics*

Description

`phy_qc()` returns a tibble containing important metrics about the sparsity and sequencing depth of the count matrix. These metrics are calculated both globally and split by the levels of the categorical variable of interest. The content of the table is as follows:

Usage

```
phy_qc(rec)
```

Arguments

`rec`	A `Recipe` object.

Details

- `var_levels`: levels of the categorical variable of interest. "all" refers to all rows of the dataset (without splitting by categorical levels).
- `n`: total number of values in the count matrix.
- `n_zero`: number of zeros in the count matrix.
- `pct_zero`: percentage of zeros in the count matrix.
- `pct_all_zero`: percentage of taxa with zero counts in all samples.
- `pct_singletons`: percentage of taxa with counts in a single sample.
- `pct_doubletons`: percentage of taxa with counts in exactly two samples.
- `n_samples`: total number of samples in the group.
- `lib_size_min`: minimum library size (total counts) across samples in the group.
- `lib_size_max`: maximum library size (total counts) across samples in the group.
- `count_mean`: average of the mean counts per sample.
- `count_max`: average of the max counts per sample.

Value

A tibble with QC metrics.

Examples

```
data(metaHIV_phy)

## 1. Init Recipe
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")

## 2. Get QC metrics
phy_qc(rec)
```

---

prep *Performs all the steps defined in a Recipe*

Description

For a Recipe with at least one preprocessing or DA operation run the steps in a convenient order.

Usage

```
prep(
  rec,
  parallel = TRUE,
  workers = future::availableCores(constraints = "connections-16"),
  force = FALSE
)
```

Arguments

rec	A Recipe object.
parallel	if FALSE, no palatalization. if TRUE, parallel execution using future and furrr packages.
workers	Number of workers for palatalization.
force	Force the reexecution of all steps. This remove previous results.

Value

A PrepRecipe object.

Examples

```
data(metaHIV_phy)
rec <-
  recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Class") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = function(x) sum(x > 0) >= (0.03 * length(x))) |>
  step_maaslin()

da_results <- prep(rec)
```

PrepRecipe-class *PrepRecipe-class object*

Description

A PrepRecipe is Recipe with the results corresponding to the steps defined in the Recipe.

Value

PrepRecipe-class object

Slots

`results` Contains the results of all defined analysis in the Recipe.

`bakes` Contains the executed bakes.

prep_recipe *Create a PrepRecipe.*

Description

A PrepRecipe is Recipe with the results corresponding to the steps defined in the Recipe.

Usage

```
prep_recipe(rec, results, bakes)
```

Arguments

`rec`	A Recipe object.
`results`	list with the results
`bakes`	list with saved bakes

Value

An object of class `PrepRecipe`.

rand_id *Make a random identification field for steps*

Description

Make a random identification field for steps

Usage

```
rand_id(prefix = "step")
```

Arguments

prefix	A single character string

Value

A character string with the prefix and random letters separated by and underscore.

Examples

```
rand_id("step")
```

rarefaction_help *Information about the Rarefaction process*

Description

Prints detailed information regarding the use of rarefaction in microbiome analysis, its implications, and relevant literature.

Usage

```
rarefaction_help()
```

Value

invisible

Examples

```
rarefaction_help()
```

recipe *Create a Recipe for preprocessing data*

Description

A Recipe is a description of the steps to be applied to a data set in order to prepare it for data analysis.

Usage

```
recipe(
  microbiome_object = NULL,
  var_info = NULL,
  tax_info = NULL,
  steps = list()
)
```

Arguments

microbiome_object	Phyloseq-class object or TreeSummarizedExperiment-class object.
var_info	A character string of column names corresponding to variables that will be used in any context.
tax_info	A character string of taxonomic levels that will be used in any context.
steps	list with steps.

Value

An object of class `Recipe` with sub-objects:

phyloseq	object of class phyloseq with taxa abundance information.
var_info	A tibble that contains the current set of terms in the data set. This initially defaults to the same data contained in `var_info`.
tax_info	A tibble that contains the current set of taxonomic levels that will be used in the analysis.

Examples

```
data(metaHIV_phy)

## Define recipe
rec <-
  recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.3 * length(x))") |>
  step_deseq() |>
  step_maaslin()
```

```
## Prep recipe
da_results <- prep(rec)

## Consensus strategy
n_methods <- 2
da_results <- bake(da_results, count_cutoff = n_methods)

## Results
cool(da_results)

## You can also crate a recipe without var and tax info
rec <- recipe(metaHIV_phy)

rec

## And define them later
rec <- rec |>
  add_var("RiskGroup2") |>
  add_tax("Genus")

rec

## When trying to add an identical step to an existing one, the system
## returns an information message.
rec <- step_aldex(rec)
rec <- step_aldex(rec)

## The same with bake
da_results <- bake(da_results)
da_results <- bake(da_results)
```

`recipes_pkg_check` *Update packages*

Description

This will check to see if all required packages are installed.

Usage

```
recipes_pkg_check(pkg = NULL, step_name, ...)
```

Arguments

`pkg`	A character string for the package being checked
`step_name`	Name of the step.
`...`	Extra arguments to pass to `utils::install.packages()`

Value

Nothing is returned but a message is printed to the console about which packages (if any) should be installed along with code to do so.

`required_deps` *Returns required pakcages for Recipe object*

Description

Returns required pakcages for Recipe object

Usage

```
required_deps(rec)
```

Arguments

`rec`	A `Recipe` object

Value

character

Examples

```
data(test_rec)

## The function returns instructions to install any uninstalled dependencies
## needed to run the Recipe steps
dar:::required_deps(test_rec)

## The function also works with PrepRecipe-class objects
data(test_prep_rec)
dar:::required_deps(test_prep_rec)
```

`sample_data` *Extracts sample_data from phyloseq inside a Recipe*

Description

Extracts sample_data from phyloseq inside a Recipe

Usage

```
sample_data(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

A tibble

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
sample_data(rec)
```

---

step *Overall Wrappers to Make New* step_X *or* check_Y *Objects*

Description

step sets the class of the step and check is for checks.

Usage

```
step(subclass, ..., .prefix = "step_")

check(subclass, ..., .prefix = "check_")
```

Arguments

subclass	A character string for the resulting class. For example, if subclass = "blah" the step object that is returned has class step_blah or check_blah depending on the context.
...	All arguments to the operator that should be returned.
.prefix	Prefix to the subclass created.

Value

An updated step or check with the new class.

steps_ids *Get step_ids from recipe*

Description

Get step_ids from recipe

Usage

```
steps_ids(rec, type = "all")
```

Arguments

rec	A Recipe object.
type	character vector indicating the type class. Options `c("all", "da", "prepro")`.

Value

character vector

Examples

```
data(test_rec)

## We can extract the step identifiers from a Recipe with `step_ids`
ids <- steps_ids(test_rec)
ids

## With the `type` parameter, extract the prepro and da steps separately.
da_ids <- steps_ids(test_rec, type = "da")
da_ids

prepro_ids <- steps_ids(test_rec, type = "prepro")
prepro_ids
```

step_aldex *ALDEx2 analysis*

Description

A differential abundance analysis for the comparison of two or more conditions. For example, single-organism and meta-RNA-seq high-throughput sequencing assays, or of selected and unselected values from in-vitro sequence selections. Uses a Dirichlet-multinomial model to infer abundance from counts, that has been optimized for three or more experimental replicates. Infers sampling variation and calculates the expected false discovery rate given the biological and sampling variation using the Wilcox rank test or Welches t-test (aldex.ttest) or the glm and Kruskal Wallis tests (aldex.glm). Reports both P and fdr values calculated by the Benjamini Hochberg correction (Not supported in dar package).

Usage

```
step_aldex(
  rec,
  max_significance = 0.05,
  mc.samples = 128,
  denom = "all",
  rarefy = FALSE,
  id = rand_id("aldex")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
max_significance	Benjamini-Hochberg corrected P value of Welch's t test cutoff.
mc.samples	The number of Monte Carlo instances to use to estimate the underlying distributions; since we are estimating central tendencies, 128 is usually sufficient, but larger numbers may be .
denom	An any variable (all, iqlr, zero, lvha, median, user) indicating features to use as the denominator for the Geometric Mean calculation The default "all" uses the geometric mean abundance of all features. Using "median" returns the median abundance of all features. Using "iqlr" uses the features that are between the first and third quartile of the variance of the clr values across all samples. Using "zero" uses the non-zero features in each grop as the denominator. This approach is an extreme case where there are many nonzero features in one condition but many zeros in another. Using "lvha" uses features that have low variance (bottom quartile) and high relative abundance (top quartile in every sample). It is also possible to supply a vector of row indices to use as the denominator. Here, the experimentalist is determining a-priori which rows are thought to be invariant. In the case of RNA-seq, this could include ribosomal protein genes and and other house-keeping genes. This should be used with caution because the offsets may be different in the original data and in the data used by the function because features that are 0 in all samples are removed by aldex.clr.
rarefy	Boolean indicating if OTU counts must be rarefyed. This rarefaction uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. If 'no_seed', rarefaction is performed without a set seed.
id	A character string that is unique to this step to identify it.

Details

The run_aldex function is a wrapper that performs log-ratio transformation and statistical testing in a single line of code. Specifically, this function: (a) generates Monte Carlo samples of the Dirichlet

distribution for each sample, (b) converts each instance using a log-ratio transform, then (c) returns test results for two sample (Welch's t, Wilcoxon) test. This function also estimates effect size for two sample analyses.

Value

An object of class `Recipe`

See Also

Other Diff taxa steps: `step_ancom()`, `step_corncob()`, `step_deseq()`, `step_lefse()`, `step_maaslin()`, `step_wilcox()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.4 * length(x))")

rec

## Define ALDEX step with default parameters and prep
rec <-
  step_aldex(rec) |>
  prep(parallel = FALSE)

rec

## Wearing rarefaction only for this step
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  step_aldex(rarefy = TRUE)

rec
```

---

step_ancom *ANCOM analysis*

Description

Determine taxa whose absolute abundances, per unit volume, of the ecosystem (e.g., gut) are significantly different with changes in the covariate of interest (e.g., group). The current version of ancombc2 function implements Analysis of Compositions of Microbiomes with Bias Correction (ANCOM-BC2) in cross-sectional and repeated measurements data. In addition to the two-group comparison, ANCOM-BC2 also supports testing for continuous covariates and multi-group comparisons, including the global test, pairwise directional test, Dunnett's type of test, and trend test.

Usage

```
step_ancom(
  rec,
  fix_formula = get_var(rec)[[1]],
  rand_formula = NULL,
  p_adj_method = "holm",
  prv_cut = 0.1,
  lib_cut = 0,
  s0_perc = 0.05,
  group = NULL,
  struc_zero = FALSE,
  neg_lb = FALSE,
  alpha = 0.05,
  n_cl = 1,
  verbose = FALSE,
  global = FALSE,
  pairwise = FALSE,
  dunnet = FALSE,
  trend = FALSE,
  rarefy = FALSE,
  id = rand_id("ancom")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
fix_formula	the character string expresses how the microbial absolute abundances for each taxon depend on the fixed effects in metadata. When specifying the fix_formula, make sure to include the group variable in the formula if it is not NULL.
rand_formula	the character string expresses how the microbial absolute abundances for each taxon depend on the random effects in metadata. ANCOM-BC2 follows the lmerTest package in formulating the random effects. See ?lmerTest::lmer for more details. Default is NULL.
p_adj_method	character. method to adjust p-values. Default is "holm". Options include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See ?stats::p.adjust for more details.
prv_cut	a numerical fraction between 0 and 1. Taxa with prevalences less than prv_cut will be excluded in the analysis. For instance, suppose there are 100 samples, if a taxon has nonzero counts presented in less than 10 samples, it will not be further analyzed. Default is 0.10.
lib_cut	a numerical threshold for filtering samples based on library sizes. Samples with library sizes less than lib_cut will be excluded in the analysis. Default is 0, i.e. do not discard any sample.
s0_perc	a numerical fraction between 0 and 1. Inspired by Significance Analysis of Microarrays (SAM) methodology, a small positive constant is added to the denominator of ANCOM-BC2 test statistic corresponding to each taxon to avoid

	the significance due to extremely small standard errors, especially for rare taxa. This small positive constant is chosen as s0_perc-th percentile of standard error values for each fixed effect. Default is 0.05 (5th percentile).
group	character. The name of the group variable in metadata. group should be discrete. Specifying group is required for detecting structural zeros and performing multi-group comparisons (global test, pairwise directional test, Dunnett's type of test, and trend test). Default is NULL. If the group of interest contains only two categories, leave it as NULL.
struc_zero	logical. Whether to detect structural zeros based on group. Default is FALSE. See Details for a more comprehensive discussion on structural zeros.
neg_lb	logical. Whether to classify a taxon as a structural zero using its asymptotic lower bound. Default is FALSE.
alpha	numeric. Level of significance. Default is 0.05.
n_cl	numeric. The number of nodes to be forked. For details, see ?parallel::makeCluster. Default is 1 (no parallel computing).
verbose	logical. Whether to generate verbose output during the ANCOM-BC2 fitting process. Default is FALSE.
global	logical. Whether to perform the global test. Default is FALSE.
pairwise	logical. Whether to perform the pairwise directional test. Default is FALSE.
dunnet	logical. Whether to perform the Dunnett's type of test. Default is FALSE.
trend	logical. Whether to perform trend test. Default is FALSE.
rarefy	Boolean indicating if OTU counts must be rarefyed. This rarefaction uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. If 'no_seed', rarefaction is performed without a set seed.
id	A character string that is unique to this step to identify it.

Value

An object of class `Recipe`

See Also

Other Diff taxa steps: `step_aldex()`, `step_corncob()`, `step_deseq()`, `step_lefse()`, `step_maaslin()`, `step_wilcox()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Genus") |>
```

```
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.4 * length(x))")

rec

## Define step with default parameters and prep
rec <-
  step_ancom(rec) |>
  prep(parallel = FALSE)

rec

## Wearing rarefaction only for this step
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  step_ancom(rarefy = TRUE)

rec
```

---

step_corncob *corncob analysis*

Description

Corncob an individual taxon regression model that uses abundance tables and sample data. corncob is able to model differential abundance and differential variability, and addresses each of the challenges presented below:

Usage

```
step_corncob(
  rec,
  phi.formula = stats::formula(~1),
  formula_null = stats::formula(~1),
  phi.formula_null = stats::formula(~1),
  link = "logit",
  phi.link = "logit",
  test = "Wald",
  boot = FALSE,
  B = 1000,
  filter_discriminant = TRUE,
  fdr_cutoff = 0.05,
  fdr = "fdr",
  log2FC = 0,
  rarefy = FALSE,
  id = rand_id("corncob")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
phi.formula	An object of class formula without the response: a symbolic description of the model to be fitted to the dispersion.
formula_null	Formula for mean under null, without response.
phi.formula_null	Formula for overdispersion under null, without response.
link	Link function for abundance covariates, defaults to "logit".
phi.link	Link function for dispersion covariates, defaults to "logit".
test	Character. Hypothesis testing procedure to use. One of "Wald" or "LRT" (likelihood ratio test).
boot	Boolean. Defaults to FALSE. Indicator of whether or not to use parametric bootstrap algorithm. (See pbWald and pbLRT).
B	Optional integer. Number of bootstrap iterations. Ignored if boot is FALSE. Otherwise, defaults to 1000.
filter_discriminant	Boolean. Defaults to TRUE. If FALSE, discriminant taxa will not be filtered out.
fdr_cutoff	Integer. Defaults to 0.05. Desired type 1 error rate.
fdr	Character. Defaults to "fdr". False discovery rate control method, see p.adjust for more options.
log2FC	log2FC cutoff.
rarefy	Boolean indicating if OTU counts must be rarefyed. This rarefaction uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. If 'no_seed', rarefaction is performed without a set seed.
id	A character string that is unique to this step to identify it.

Details

- different sequencing depth
- excessive zeros from unobserved taxa
- high variability of empirical relative abundances (overdispersion)
- within-taxon correlation
- hypothesis testing with categorical and continuous covariates

Value

An object of class Recipe

See Also

Other Diff taxa steps: step_aldex(), step_ancom(), step_deseq(), step_lefse(), step_maaslin(), step_wilcox()

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.3 * length(x))")

rec

## Define step with default parameters and prep
rec <-
  step_corncob(rec) |>
  prep(parallel = FALSE)

rec
```

---

step_deseq *DESeq2 analysis*

Description

Differential expression analysis based on the Negative Binomial (a.k.a. Gamma-Poisson) distribution. This function performs a default analysis through the steps: 1) estimation of size factors: estimateSizeFactors. 2) estimation of dispersion: estimateDispersions. 3) Negative Binomial GLM fitting and Wald statistics: nbinomWaldTest. For complete details on each step, see the manual pages of the respective functions. After the DESeq function returns a DESeqDataSet object, results tables (log2 fold changes and p-values) can be generated using the results function. Shrunken LFC can then be generated using the lfcShrink function.

Usage

```
step_deseq(
  rec,
  test = "Wald",
  fitType = "local",
  betaPrior = FALSE,
  type = "ashr",
  max_significance = 0.05,
  log2FC = 0,
  rarefy = FALSE,
  id = rand_id("deseq")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
test	Either "Wald" or "LRT", which will then use either Wald significance tests (defined by nbinomWaldTest), or the likelihood ratio test on the difference in deviance between a full and reduced model formula (defined by nbinomLRT).
fitType	either "parametric", "local", "mean", or "glmGamPoi" for the type of fitting of dispersions to the mean intensity. See estimateDispersions for description.
betaPrior	whether or not to put a zero-mean normal prior on the non-intercept coefficients See nbinomWaldTest for description of the calculation of the beta prior. In versions >=1.16, the default is set to FALSE, and shrunken LFCs are obtained afterwards using lfcShrink.
type	"apeglm" is the adaptive Student's t prior shrinkage estimator from the 'apeglm' package; "ashr" is the adaptive shrinkage estimator from the 'ashr' package, using a fitted mixture of normals prior - see the Stephens (2016) reference below for citation; "normal" is the 2014 DESeq2 shrinkage estimator using a Normal prior.
max_significance	The q-value threshold for significance.
log2FC	log2FC cutoff.
rarefy	Boolean indicating if OTU counts must be rarefyed. This rarefaction uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. If 'no_seed', rarefaction is performed without a set seed.
id	A character string that is unique to this step to identify it.

Value

An object of class `Recipe`

See Also

Other Diff taxa steps: `step_aldex()`, `step_ancom()`, `step_corncob()`, `step_lefse()`, `step_maaslin()`, `step_wilcox()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.4 * length(x))")
```

```
rec

## Define step with default parameters and prep
rec <-
  step_deseq(rec) |>
  prep(parallel = FALSE)

rec

## Wearing rarefaction only for this step
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  step_deseq(rarefy = TRUE)

rec
```

`step_filter_by_abundance` *Filter taxa by abundance*

Description

This is a convenience wrapper around the filter_taxa function. It is intended to speed up filtering complex experimental objects with one function call. In the case of filter_by_abundance, the filtering will be based on the relative abundance of each taxon. The taxa retained in the dataset are those where the sum of their abundance is greater than the product of the total abundance and the provided threshold.

Usage

```
step_filter_by_abundance(
  rec,
  threshold = 0.01,
  id = rand_id("filter_by_abundance")
)
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this Recipe.
`threshold`	The relative abundance threshold for filtering taxa, expressed as a proportion of the total abundance. For example, a threshold of 0.01 means that a taxon must make up at least 1% of the total abundance to be retained. The default value is 0.01.
`id`	A character string that is unique to this step to identify it.

Details

The function calculates the total abundance of all taxa in the phyloseq object. It then compares this total abundance to the abundance of each individual taxon. If a taxon's abundance is less than the threshold times the total abundance, that taxon is removed from the phyloseq object.

Value

A Recipe object that has been filtered based on abundance.

Note

This function modifies `rec` in place, you might want to make a copy of `rec` before modifying it if you need to preserve the original object.

See Also

`filter_taxa`

Other filter phy steps: `step_filter_by_prevalence()`, `step_filter_by_rarity()`, `step_filter_by_variance()`, `step_filter_taxa()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Phylum")
rec

## Define filter_by_abundance step with default parameters
rec <- step_filter_by_abundance(rec, threshold = 0.01)
rec
```

---

step_filter_by_prevalence

Filter taxa by prevalence

Description

This is a convenience function around the filter_taxa function. It is designed to speed up filtering complex experimental objects with one function call. In the case of run_filter_by_prevalence, the filtering will be based on the prevalence of each taxon. The taxa retained in the dataset are those where the prevalence is greater than the provided threshold.

Usage

```
step_filter_by_prevalence(
  rec,
  threshold = 0.01,
  id = rand_id("filter_by_prevalence")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
threshold	The prevalence threshold for filtering taxa, expressed as a proportion of the total number of samples. For example, a threshold of 0.01 means that a taxon must be present in at least 1% of the samples to be retained. The default value is 0.01.
id	A character string that is unique to this step to identify it.

Details

The function calculates the prevalence of all taxa in the phyloseq object as the proportion of samples in which they are present. It then compares this prevalence to the threshold. If a taxon's prevalence is less than the threshold, that taxon is removed from the phyloseq object.

Value

A Recipe object that has been filtered based on prevalence.

Note

This function modifies `rec` in place, you might want to make a copy of `rec` before modifying it if you need to preserve the original object.

See Also

`filter_taxa`

Other filter phy steps: `step_filter_by_abundance()`, `step_filter_by_rarity()`, `step_filter_by_variance()`, `step_filter_taxa()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Phylum")
rec

## Define step_filter_by_prevalence step with default parameters
rec <- step_filter_by_prevalence(rec, threshold = 0.01)
rec
```

`step_filter_by_rarity` *Filter taxa by rarity*

Description

This is a convenience function around the filter_taxa function. It is designed to speed up filtering complex experimental objects with one function call. In the case of run_filter_by_rarity, the filtering will be based on the rarity of each taxon. The taxa retained in the dataset are those where the prevalence (proportion of samples where present) is LESS than the provided threshold.

Usage

```
step_filter_by_rarity(rec, threshold = 0.01, id = rand_id("filter_by_rarity"))
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this Recipe.
`threshold`	The rarity threshold for filtering taxa, expressed as a proportion of the total number of samples. For example, a threshold of 0.01 means that a taxon must be present in less than 1% of the samples to be retained. The default value is 0.01.
`id`	A character string that is unique to this step to identify it.

Details

The function calculates the prevalence of all taxa in the phyloseq object as the proportion of samples in which they are present. It then compares this prevalence to the threshold. If a taxon's prevalence is greater than or equal to the threshold, that taxon is removed (filtered out) from the phyloseq object, leaving only the "rare" taxa.

Value

A Recipe object that has been filtered based on rarity.

Note

This function modifies `rec` in place, you might want to make a copy of `rec` before modifying it if you need to preserve the original object.

See Also

`filter_taxa`

Other filter phy steps: `step_filter_by_abundance()`, `step_filter_by_prevalence()`, `step_filter_by_variance()`, `step_filter_taxa()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Phylum")
rec

## Define step_filter_by_rarity step with default parameters
rec <- step_filter_by_rarity(rec, threshold = 0.01)
rec
```

`step_filter_by_variance` *Filter taxa by variance*

Description

This is a convenience function around the filter_taxa function. It is designed to speed up filtering complex experimental objects with one function call. In the case of run_filter_by_variance, the filtering will be based on the variance of each taxon. The taxa retained in the dataset are those where the variance of their abundance is greater than the provided threshold.

Usage

```
step_filter_by_variance(
  rec,
  threshold = 0.01,
  id = rand_id("filter_by_variance")
)
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this Recipe.
`threshold`	The variance threshold for filtering taxa. The default value is 0.01.
`id`	A character string that is unique to this step to identify it.

Details

The function calculates the variance of all taxa in the phyloseq object. It then compares this variance to the variance of each individual taxon. If a taxon's variance is less than or equal to the threshold, that taxon is removed from the phyloseq object.

Value

A Recipe object that has been filtered based on variance.

Note

This function modifies `rec` in place, you might want to make a copy of `rec` before modifying it if you need to preserve the original object.

See Also

`filter_taxa`

Other filter phy steps: `step_filter_by_abundance()`, `step_filter_by_prevalence()`, `step_filter_by_rarity()`, `step_filter_taxa()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Phylum")
rec

## Define step_filter_by_variance step with default parameters
rec <- step_filter_by_variance(rec, threshold = 0.01)
rec
```

---

step_filter_taxa *Filter taxa based on across-sample OTU abundance criteria*

Description

This function is directly analogous to the genefilter function for microarray filtering, but is used for filtering OTUs from phyloseq objects. It applies an arbitrary set of functions — as a function list, for instance, created by filterfun — as across-sample criteria, one OTU at a time. It takes as input a phyloseq object, and returns a logical vector indicating whether or not each OTU passed the criteria. Alternatively, if the "prune" option is set to FALSE, it returns the already-trimmed version of the phyloseq object.

Usage

```
step_filter_taxa(rec, .f, id = rand_id("filter_taxa"))
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this Recipe.
`.f`	A function or list of functions that take a vector of abundance values and return a logical. Some canned useful function types are included in the genefilter-package.
`id`	A character string that is unique to this step to identify it.

Value

An object of class Recipe

See Also

Other filter phy steps: step_filter_by_abundance(), step_filter_by_prevalence(), step_filter_by_rarity(), step_filter_by_variance()

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Phylum")
rec

## Define filter taxa step with default parameters
rec <-
  step_filter_taxa(rec, .f = "function(x) sum(x > 0) >= (0.03 * length(x))")

rec
```

---

step_lefse *lefse analysis*

Description

Lefser is metagenomic biomarker discovery tool that is based on LEfSe tool and is published by Huttenhower et al. 2011. Lefser is the R implementation of the LEfSe method. Using statistical analyses, lefser compares microbial populations of healthy and diseased subjects to discover differencially expressed microorganisms. Lefser than computes effect size, which estimates magnitude of differential expression between the populations for each differentially expressed microorganism. Subclasses of classes can also be assigned and used within the analysis.

Usage

```
step_lefse(
  rec,
  kruskal.threshold = 0.05,
  wilcox.threshold = 0.05,
  lda.threshold = 2,
  subclassCol = NULL,
  assay = 1L,
  trim.names = FALSE,
  rarefy = TRUE,
  id = rand_id("lefse")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
kruskal.threshold	numeric(1) The p-value for the Kruskal-Wallis Rank Sum Test (default 0.05).
wilcox.threshold	numeric(1) The p-value for the Wilcoxon Rank-Sum Test when 'blockCol' is present (default 0.05).
lda.threshold	numeric(1) The effect size threshold (default 2.0).
subclassCol	character(1) Optional column name in 'colData(expr)' indicating the blocks, usually a factor with two levels (e.g., 'c("adult", "senior")'; default NULL).
assay	The i-th assay matrix in the 'SummarizedExperiment' ('expr'; default 1).
trim.names	If 'TRUE' extracts the most specific taxonomic rank of organism.
rarefy	Boolean indicating if OTU counts must be rarefyed. This rarefaction uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. If 'no_seed', rarefaction is performed without a set seed.
id	A character string that is unique to this step to identify it.

Value

An object of class `Recipe`

See Also

Other Diff taxa steps: `step_aldex()`, `step_ancom()`, `step_corncob()`, `step_deseq()`, `step_maaslin()`, `step_wilcox()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.3 * length(x))")

rec

## Define step with default parameters
rec <- step_lefse(rec)

rec
```

```
## Running lefse without rarefaction (not recommended)
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  step_lefse(rarefy = FALSE)

rec
```

`step_maaslin` *MaAsLin3 analysis*

Description

MaAsLin 3 finds associations between microbiome meta-omics features and complex metadata. It uses a unified framework to test for both abundance (using linear models) and prevalence (using logistic regression).

Usage

```
step_maaslin(
  rec,
  min_abundance = 0,
  min_prevalence = 0.1,
  min_variance = 0,
  normalization = "TSS",
  transform = "LOG",
  max_significance = 0.1,
  random_effects = NULL,
  correction = "BH",
  standardize = TRUE,
  reference = NULL,
  median_comparison_abundance = TRUE,
  rarefy = FALSE,
  id = rand_id("maaslin")
)
```

Arguments

`rec`	A Recipe object. The step will be added to the sequence of operations for this Recipe.
`min_abundance`	The minimum abundance for each feature.
`min_prevalence`	The minimum percent of samples for which a feature is detected at minimum abundance.
`min_variance`	Keep features with variance greater than.
`normalization`	The normalization method to apply. Default: "TSS". Choices: "TSS", "CLR", "NONE".

transform	The transform to apply. Default: "LOG" (Base 2). Choices: "LOG", "PLOG", "NONE".
max_significance	The q-value threshold for significance.
random_effects	The random effects for the model (vector of character strings).
correction	The correction method for computing the q-value.
standardize	Apply z-score so continuous metadata are on the same scale.
reference	The factor to use as a reference for a variable with more than two levels provided as a string of 'variable,reference' semi-colon delimited.
median_comparison_abundance	Test abundance coefficients against a null value corresponding to the median coefficient for a metadata variable across the features. Recommended for relative abundance (default: TRUE).
rarefy	Boolean indicating if OTU counts must be rarefyed.
id	A character string that is unique to this step to identify it.

Value

An object of class Recipe

See Also

Other Diff taxa steps: step_aldex(), step_ancom(), step_corncob(), step_deseq(), step_lefse(), step_wilcox()

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = "function(x) sum(x > 0) >= (0.4 * length(x))")

rec

## Define step with default parameters and prep
rec <-
  step_maaslin(rec) |>
  prep(parallel = FALSE)

rec

## Wearing rarefaction only for this step
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Species") |>
  step_maaslin(rarefy = TRUE)

rec
```

step_metagenomeseq *(DEFUNCT) Zero-inflated Log-Normal mixture model for differential abundance*

Description

WARNING: This function is **DEFUNCT** and has been removed from the package. It is no longer maintained.

Usage

```
step_metagenomeseq(rec, ...)
```

Arguments

rec	A Recipe object.
...	Any other arguments (ignored).

step_rarefaction *Resample an OTU table such that all samples have the same library size.*

Description

Please note that the authors of phyloseq do not advocate using this as a normalization procedure, despite its recent popularity. Our justifications for using alternative approaches to address disparities in library sizes have been made available as an article in PLoS Computational Biology. See phyloseq_to_deseq2 for a recommended alternative to rarefying directly supported in the phyloseq package, as well as the supplemental materials for the PLoS-CB article and the phyloseq extensions repository on GitHub. Nevertheless, for comparison and demonstration, the rarefying procedure is implemented here in good faith and with options we hope are useful. This function uses the standard R sample function to resample from the abundance values in the otu_table component of the first argument, physeq. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the sample.size is expected. This kind of resampling can be performed with and without replacement, with replacement being the more computationally-efficient, default setting. See the replace parameter documentation for more details. We recommended that you explicitly select a random number generator seed before invoking this function, or, alternatively, that you explicitly provide a single positive integer argument as rngseed.

Usage

```
step_rarefaction(rec, id = rand_id("rarefaction"))
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
id	A character string that is unique to this step to identify it.

Value

An object of class Recipe

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea")) |>
  step_filter_taxa(.f = function(x) sum(x > 0) >= (0.03 * length(x)))

rec

## Define step with default parameters and prep
rec <- step_rarefaction(rec)

rec
```

---

step_subset_taxa *Subset taxa by taxonomic level*

Description

This is a convenience function around the subset_taxa function from the phyloseq package. It is designed to speed up subsetting complex experimental objects with one function call. In the case of run_subset_taxa, the subsetting will be based on the taxonomic level of each taxon. The taxa retained in the dataset are those where the taxonomic level matches the provided taxa.

Usage

```
step_subset_taxa(rec, tax_level, taxa, id = rand_id("subset_taxa"))
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
tax_level	The taxonomic level for subsetting taxa.
taxa	The taxa to be retained in the dataset.
id	A character string that is unique to this step to identify it.

Details

The function subsets the taxa in the phyloseq object based on the provided taxonomic level and taxa. Only the taxa that match the provided taxa at the given taxonomic level are retained in the phyloseq object.

Value

A Recipe object that has been subsetted based on taxonomic level.

Note

This function modifies `rec` in place, you might want to make a copy of `rec` before modifying it if you need to preserve the original object.

See Also

`subset_taxa`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <- recipe(metaHIV_phy, "RiskGroup2", "Species")
rec

## Define step_subset_taxa step with default parameters
rec <- step_subset_taxa(
  rec,
  tax_level = "Kingdom",
  taxa = c("Bacteria", "Archaea")
)
prep(rec)
```

---

step_wilcox *Wilcox analysis*

Description

Performs a wilcox test to determine features (be it Operational Taxanomic Unit (OTU), species, etc.) that are differentially abundant between two or more groups of multiple samples.

Usage

```
step_wilcox(
  rec,
  norm_method = "compositional",
  max_significance = 0.05,
```

```
  p_adj_method = "BH",
  rarefy = FALSE,
  id = rand_id("wilcox")
)
```

Arguments

rec	A Recipe object. The step will be added to the sequence of operations for this Recipe.
norm_method	Transformation to apply. The options include: 'compositional' (ie relative abundance), 'Z', 'log10', 'log10p', 'hellinger', 'identity', 'clr', 'alr', or any method from the vegan::decostand function.
max_significance	The q-value threshold for significance.
p_adj_method	Character. Specifying the method to adjust p-values for multiple comparisons. Default is "BH" (Benjamini-Hochberg procedure).
rarefy	Boolean indicating if OTU counts must be rarefyed.
id	A character string that is unique to this step to identify it.

Value

An object of class `Recipe`

See Also

Other Diff taxa steps: `step_aldex()`, `step_ancom()`, `step_corncob()`, `step_deseq()`, `step_lefse()`, `step_maaslin()`

Examples

```
data(metaHIV_phy)

## Init Recipe
rec <-
  recipe(metaHIV_phy, "RiskGroup2", "Phylum") |>
  step_subset_taxa(tax_level = "Kingdom", taxa = c("Bacteria", "Archaea"))

## Define step with default parameters
rec <- step_wilcox(rec)
rec
```

tax_table *Extracts tax_table from phyloseq inside a Recipe*

Description

Extracts tax_table from phyloseq inside a Recipe

Usage

```
tax_table(rec)
```

Arguments

rec	A Recipe or PrepRecipe object.

Value

A tibble

Examples

```
data(metaHIV_phy)
rec <- recipe(metaHIV_phy, var_info = "RiskGroup2", tax_info = "Species")
tax_table(rec)
```

test_prep_rec *PrepRecipe for metaHIV_phy data*

Description

A Recipe created for a metaHIV_phy object uning "Riskgroup2" as a var_info and "Genus" as a tax_info. Also includes step_deseq and step_maaslin.

Usage

```
data("test_prep_rec")
```

Format

A PrepRecipe object.

test_rec *Recipe for metaHIV_phy data*

Description

A Recipe created for a metaHIV_phy object uning "Riskgroup2" as a `var_info` and "Genus" as a `tax_info`.

Usage

```
data("test_rec")
```

Format

A Recipe object.

tidyeval *Tidy eval helpers*

Description

This page lists the tidy eval tools reexported in this package from rlang. To learn about using tidy eval in scripts and packages at a high level, see the dplyr programming vignette and the ggplot2 in packages vignette. The Metaprogramming section of Advanced R may also be useful for a deeper dive.

- The tidy eval operators `{{`, `!!`, and `!!!` are syntactic constructs which are specially interpreted by tidy eval functions. You will mostly need `{{`, as `!!` and `!!!` are more advanced operators which you should not have to use in simple cases.

  The curly-curly operator `{{` allows you to tunnel data-variables passed from function arguments inside other tidy eval functions. `{{` is designed for individual arguments. To pass multiple arguments contained in dots, use `...` in the normal way.

  ```
  my_function <- function(data, var, ...) {
    data %>%
      group_by(...) %>%
      summarise(mean = mean({{ var }}))
  }
  ```

- `enquo()` and `enquos()` delay the execution of one or several function arguments. The former returns a single expression, the latter returns a list of expressions. Once defused, expressions will no longer evaluate on their own. They must be injected back into an evaluation context with `!!` (for a single expression) and `!!!` (for a list of expressions).

```
my_function <- function(data, var, ...) {
  # Defuse
  var <- enquo(var)
  dots <- enquos(...)

  # Inject
  data %>%
    group_by(!!!dots) %>%
    summarise(mean = mean(!!var))
}
```

In this simple case, the code is equivalent to the usage of `{{` and `...` above. Defusing with `enquo()` or `enquos()` is only needed in more complex cases, for instance if you need to inspect or modify the expressions in some way.

- The `.data` pronoun is an object that represents the current slice of data. If you have a variable name in a string, use the `.data` pronoun to subset that variable with `[[`.

```
my_var <- "disp"
mtcars %>% summarise(mean = mean(.data[[my_var]]))
```

- Another tidy eval operator is `:=`. It makes it possible to use glue and curly-curly syntax on the LHS of `=`. For technical reasons, the R language doesn't support complex expressions on the left of `=`, so we use `:=` as a workaround.

```
my_function <- function(data, var, suffix = "foo") {
  # Use `{{` to tunnel function arguments and the usual glue
  # operator `{` to interpolate plain strings.
  data %>%
    summarise("{{ var }}_mean_{suffix}" := mean({{ var }}))
}
```

- Many tidy eval functions like `dplyr::mutate()` or `dplyr::summarise()` give an automatic name to unnamed inputs. If you need to create the same sort of automatic names by yourself, use `as_label()`. For instance, the glue-tunnelling syntax above can be reproduced manually with:

```
my_function <- function(data, var, suffix = "foo") {
  var <- enquo(var)
  prefix <- as_label(var)
  data %>%
    summarise("{prefix}_mean_{suffix}" := mean(!!var))
}
```

Expressions defused with `enquo()` (or tunnelled with `{{`) need not be simple column names, they can be arbitrarily complex. `as_label()` handles those cases gracefully. If your code assumes a simple column name, use `as_name()` instead. This is safer because it throws an error if the input is not a name as expected.

Value

The function does not return a value explicitly.

Examples

```
# `enquo()` defuses the expression supplied by your user
f <- function(arg) {
  rlang::enquo(arg)
}

f(1 + 1)

# `enquos()` works with arguments and dots. It returns a list of
# expressions
f <- function(...) {
  rlang::enquos(...)
}

f(1 + 1, 2 * 10)

# Let's create some symbols:
foo <- quote(foo)
bar <- rlang::sym("bar")

# as_name() converts symbols to strings:
foo

rlang::as_name(foo)

typeof(bar)

typeof(rlang::as_name(bar))

# as_name() unwraps quosured symbols automatically:
rlang::as_name(rlang::quo(foo))

# as_label() is useful with quoted expressions:
rlang::as_label(rlang::expr(foo(bar)))

rlang::as_label(rlang::expr(foobar))

# It works with any R object. This is also useful for quoted
# arguments because the user might unquote constant objects:
rlang::as_label(1:3)

rlang::as_label(base::list)
```

`to_tibble` *Wrapper to convert phyloseq slots to tibble*

Description

Wrapper to convert phyloseq slots to tibble

Usage

```
to_tibble(df, id_name = "otu_id")
```

Arguments

df	output of otu_table(), sample_data() or tax_table() phyloseq functions.
id_name	Name of the new column generated from rownames

Value

tibble

Examples

```
data(test_rec)
otu_table <-
  get_phy(test_rec) |>
  phyloseq::otu_table()

dar:::to_tibble(otu_table)
```

---

use_rarefy *Perform Rarefaction on Phyloseq Object*

Description

This function performs rarefaction on a phyloseq object if the rarefy parameter is set to TRUE. Rarefaction is a process that randomly subsamples the data to a specified depth. This is done to account for differences in sequencing depth between samples. However, this process is not without controversy. Rarefaction can lead to loss of information and can also lead to false positives in differential abundance testing. For more information, see https://microbiomejournal.biomedcentral.com/articles/10.1186/s40168-019-0650-2

Usage

```
use_rarefy(phy, rarefy)
```

Arguments

phy	A phyloseq object.
rarefy	A logical value indicating whether to perform rarefaction. If 'no_seed', rarefaction is performed without a set seed. If FALSE, no rarefaction is performed.

Value

A phyloseq object after rarefaction if rarefy is TRUE or "no_seed", otherwise the original phyloseq object is returned.

Examples

```
data(metaHIV_phy)

## With seed
# phy_rarefied <- dar:::use_rarefy(metaHIV_phy, TRUE)

## Witout seed
# phy_rarefied <- dar:::use_rarefy(metaHIV_phy, "no_seed")
```

`%>%` *Pipe operator*

Description

Pipe operator

Value

The result of calling `rhs(lhs)`.

Index

∗ **Bake steps**
  bake, 7
∗ **Diff taxa steps**
  step_aldex, 30
  step_ancom, 32
  step_corncob, 35
  step_deseq, 37
  step_lefse, 45
  step_maaslin, 47
  step_wilcox, 51
∗ **datasets**
  metaHIV_phy, 18
  test_prep_rec, 53
  test_rec, 54
∗ **filter phy steps**
  step_filter_by_abundance, 39
  step_filter_by_prevalence, 40
  step_filter_by_rarity, 42
  step_filter_by_variance, 43
  step_filter_taxa, 44
∗ **internal**
  %>%, 58
  add_step, 5
  dar-package, 4
  get_comparisons, 13
  pastry_df, 21
  prep_recipe, 24
  recipes_pkg_check, 27
  required_deps, 28
  step, 29
  step_metagenomeseq, 49
  tidyeval, 54
  to_tibble, 56
  use_rarefy, 57
∗ **rarefaction phy steps**
  step_rarefaction, 49
∗ **subset phy steps**
  step_subset_taxa, 50
.data *(*tidyeval*)*, 54
.env *(*tidyeval*)*, 54
:= *(*tidyeval*)*, 54
%>%, 58

abundance_plt, 4
add_step, 5
add_tax, 6
add_var, 7
as_label *(*tidyeval*)*, 54
as_name *(*tidyeval*)*, 54

bake, 7

check *(*step*)*, 29
contains_rarefaction, 9
cool, 10
corr_heatmap, 10

dar *(*dar-package*)*, 4
dar-package, 4

enquo *(*tidyeval*)*, 54
enquo(), *54*
enquos *(*tidyeval*)*, 54
enquos(), *54*
exclusion_plt, 11
export_steps, 12

filter_taxa, *40–42*, *44*
find_intersections, 12

get_comparisons, 13
get_phy, 14
get_tax, 14
get_var, 15

import_steps, 15
intersection_df, 16
intersection_plt, 17

metaHIV_phy, 18

mutual_plt, 18

otu_table, 19
overlap_df, 20

pastry_df, 21
phy_qc, 22
phyloseq_or_null-class, 21
prep, 23
prep_recipe, 24
PrepRecipe *(*prep_recipe*)*, 24
PrepRecipe(), *6*
PrepRecipe-class, 24

rand_id, 25
rarefaction_help, 25
Recipe *(*recipe*)*, 26
recipe, 26
Recipe(), *6*
Recipe-class *(*phyloseq_or_null-class*)*, 21
recipes_pkg_check, 27
required_deps, 28

sample_data, 28
show,PrepRecipe-method *(*phyloseq_or_null-class*)*, 21
step, 29
step_aldex, 30, *34*, *37, 38*, *46*, *48*, *52*
step_ancom, *32*, 32, *37, 38*, *46*, *48*, *52*
step_corncob, *32*, *34*, 35, *38*, *46*, *48*, *52*
step_deseq, *32*, *34*, *37*, 37, *46*, *48*, *52*
step_filter_by_abundance, 39, *41, 42*, *44, 45*
step_filter_by_prevalence, *40*, 40, *42*, *44, 45*
step_filter_by_rarity, *40, 41*, 42, *44, 45*
step_filter_by_variance, *40–42*, 43, *45*
step_filter_taxa, *40–42*, *44*, 44
step_lefse, *32*, *34*, *37, 38*, 45, *48*, *52*
step_maaslin, *32*, *34*, *37, 38*, *46*, 47, *52*
step_metagenomeseq, 49
step_rarefaction, 49
step_subset_taxa, 50
step_wilcox, *32*, *34*, *37, 38*, *46*, *48*, 51
steps_ids, 30
subset_taxa, *51*

tax_table, 53
test_prep_rec, 53
test_rec, 54
tibble_or_NULL-class *(*phyloseq_or_null-class*)*, 21
tidyeval, 54
to_tibble, 56

use_rarefy, 57
utils::install.packages(), *27*