Package 'DspikeIn'

June 4, 2026

Type Package

Title Estimating Absolute Abundance from Microbial Spike-in Controls

Version 1.3.0

Description Provides a reproducible and modular workflow for absolute microbial quantification using spike-in controls. Supports both single spike-in taxa and synthetic microbial communities with user-defined spike-in volumes and genome copy numbers. Compatible with 'phyloseq' and 'TreeSummarizedExperiment' (TSE) data structures. The package implements methods for spike-in validation, preprocessing, scaling factor estimation, absolute abundance conversion, bias correction, and normalization. Facilitates downstream statistical analyses with 'DESeq2', 'edgeR', and other Bioconductor-compatible methods. Visualization tools are provided via 'ggplot2', 'ggtree', and related packages. Includes detailed vignettes, case studies, and function-level documentation to guide users through experimental design, quantification, and interpretation.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

LazyDataCompression xz

Depends R (>= 4.1.0)

Imports ape, Biostrings, data.table, DECIPHER, DESeq2, dplyr, edgeR, flextable, ggalluvial, ggnewscale, ggplot2, ggpubr, ggraph, ggrepel, ggridges, ggtree, ggtreeExtra, graphics, grDevices, igraph, limma, matrixStats, methods, microbiome, officer, grid, reshape2, patchwork, phangorn, phyloseq, randomForest, RColorBrewer, rlang, S4Vectors, scales, stats, tibble, tidyr, SummarizedExperiment, TreeSummarizedExperiment, utils, msa, xml2, ggstar

Suggests Biobase, mia, BiocGenerics, magrittr, BiocManager, cluster, devtools, DT, e1071, foreach, ggtext, intergraph, knitr, optparse, plyr, preprocessCore, qpdf, remotes, rmarkdown, ShortRead, testthat (>= 3.0.0), vegan, viridis

biocViews Microbiome, Preprocessing, QualityControl, DifferentialExpression, Normalization, Sequencing, Visualization, Phylogenetics, ExperimentalDesign, DataImport, Software

URL https://github.com/mghotbi/DspikeIn

BugReports https://github.com/mghotbi/DspikeIn/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

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

git_branch devel

git_last_commit 7bf75c0

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-06-04

Author Mitra Ghotbi [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9185-9993>),
Marjan Ghotbi [ctb] (ORCID: <https://orcid.org/0000-0003-4655-6445>)

Maintainer Mitra Ghotbi <mitra.ghotbi@gmail.com>

Contents

AcceptableRange *Acceptable Range Data*

Description

This dataset provides reference ranges and sample-level metadata for microbial spike-in performance evaluation. It includes taxonomic annotations and summary statistics used in validation and quality control workflows.

Usage

```
data(AcceptableRange)
```

Format

A data frame with the following columns:

Ecoregion_III EPA Level III ecoregion classification.

Genus Genus of the taxon.

Host_genus Host genus from which the sample was derived.

Percentage Observed spike-in percentage.

Phylum Phylum classification of the taxon.

Range Acceptable range category.

Total_Reads_spiked Number of reads matching the spike-in species.

Total_Reads_total Total number of reads per sample.

X Row identifier (optional, may be an index or artifact of data processing).

mean_abundance Mean abundance across all samples.

Value

A data frame of spike-in evaluation metrics and taxonomy annotations.

Source

Internal package dataset.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("AcceptableRange", package = "DspikeIn")
  head(AcceptableRange)
  summary(AcceptableRange$Percentage)
}
```

adjusted_prevalence *Adjust Prevalence in a Microbiome Object*

Description

Removes low-prevalence taxa from a phyloseq or TreeSummarizedExperiment object based on user-specified prevalence thresholds derived from taxa abundance statistics.

Usage

```
adjusted_prevalence(obj, method = "min", output_file = NULL)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.
method	Character. Threshold method: one of "min", "mean", "median", or "max".
output_file	Optional. Character. Path to save the adjusted object as .rds. Default is NULL, meaning no file will be saved unless explicitly provided.

Value

The adjusted object of the same class as obj.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  ## Adjust prevalence in phyloseq
  adjusted_physeq <- adjusted_prevalence(physeq_16SOTU, method = "mean")

  ## Convert to TreeSummarizedExperiment
  tse_obj <- convert_phyloseq_to_tse(physeq_16SOTU)

  ## Adjust prevalence in TSE
  adjusted_tse <- adjusted_prevalence(tse_obj, method = "median")
}
```

adjust_abundance_one_third *Adjust Abundance by a Custom Factor*

Description

This function normalizes microbial abundance data in a phyloseq or TreeSummarizedExperiment object by dividing all abundance values by a user-defined factor. It supports both data structures and preserves their internal metadata.

Usage

```
adjust_abundance_one_third(obj, factor = 3, output_file = NULL)

adjust_abundance_by_factor(obj, factor = 3, output_file = NULL)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbiome count data.
factor	A numeric value > 0 specifying the divisor applied to abundance counts. Default is 3 (historically used as the one-third normalization factor).
output_file	A character string specifying a path to save the adjusted object as .rds. Default is NULL (no file written).

Details

This function extracts the OTU table (for phyloseq) or assay matrix (for TSE), divides all abundance values by the provided factor, and reinserts the adjusted table while maintaining the full metadata structure.

Historically, the function name adjust_abundance_one_third() referred to a fixed normalization by 3. This generalized version preserves the same function name for backward compatibility and allows any user-defined divisor.

For clarity, an alias function adjust_abundance_by_factor() is provided.

Value

An adjusted object of the same class (phyloseq or TreeSummarizedExperiment), where the abundance values are divided by the specified factor.

See Also

convert_phyloseq_to_tse

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Adjust phyloseq object
  adjusted_physeq <- adjust_abundance_one_third(physeq_16SOTU, factor = 3)

  # Convert to TSE and adjust
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  adjusted_tse <- adjust_abundance_one_third(tse_16SOTU, factor = 2)

  # Using the alias
  adjusted_physeq2 <- adjust_abundance_by_factor(physeq_16SOTU, factor = 5)
}
```

alluvial_plot *Generate an Alluvial Plot for Microbiome Data*

Description

This function creates an alluvial plot based on input data, which can be either absolute or relative abundance data.

Usage

```
alluvial_plot(
  data,
  axes = NULL,
  abundance_threshold = 10000,
  fill_variable = "Phylum",
  silent = TRUE,
  abundance_type = "absolute",
  total_reads = NULL,
  top_taxa = NULL,
  facet_vars = NULL,
  text_size = 4,
  legend_ncol = 1,
  custom_colors = color_palette$MG,
  color_mapping = NULL
)
```

Arguments

data	A data frame containing abundance and categorical variables.
axes	A character vector specifying the categorical variables for the x-axis.
abundance_threshold	A numeric value specifying the minimum abundance required for an entity to be included in the plot. Default is 10000.
fill_variable	A string specifying the variable to be used for fill colors. Default is "Phylum".
silent	Logical. If TRUE, suppresses warnings. Default is TRUE.
abundance_type	A string specifying the type of abundance: "absolute" or "relative".
total_reads	Numeric, total number of reads for relative abundance calculation. Default is NULL.
top_taxa	Integer. The number of top abundant taxa to retain. Default is NULL.
facet_vars	A character vector specifying variables to facet by. Default is NULL.
text_size	Numeric, size of text labels. Default is 4.
legend_ncol	Integer, number of columns for the legend. Default is 1.
custom_colors	A named vector specifying colors for taxa. Default is color_palette$MG.
color_mapping	A named vector of colors for taxa, overriding custom_colors. Default is NULL.

Value

A ggplot2 object representing an alluvial plot.

Note

This function assumes data has already been converted to long format with an "Abundance" column.

Source

Built using ggalluvial, ggplot2, and dplyr for visualization of microbial abundance dynamics.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE) &&
    requireNamespace("phyloseq", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")
  physeq_subset <- phyloseq::subset_samples(physeq_16SOTU, Animal.type == "Frog")
  physeq_subset <- phyloseq::prune_taxa(
    phyloseq::taxa_sums(physeq_subset) > 0, physeq_subset
  )

  # Convert phyloseq object to long format
  pps_Abs <- get_long_format_data(physeq_subset)

  # Calculate total reads (illustrative)
  total_reads <- sum(pps_Abs$Abundance)
  message("Total reads in Frog subset: ", total_reads)

  # Heavy plotting step - wrapped in \donttest{} to reduce build time

    alluvial_plot_rel <- alluvial_plot(
      data = pps_Abs,
      axes = c("Env.broad.scale", "Host.genus", "Diet"),
      abundance_threshold = 0.01,
      fill_variable = "Phylum",
      abundance_type = "relative",
      top_taxa = 5,
      silent = TRUE,
      text_size = 3,
      legend_ncol = 1,
      custom_colors = DspikeIn::color_palette$cool_MG
    )
    print(alluvial_plot_rel)


  # Convert to TreeSummarizedExperiment (TSE) format
  tse_data <- convert_phyloseq_to_tse(physeq_subset)
  tse_long <- get_long_format_data(tse_data)

  # Heavy plotting step - wrapped in \donttest{} to reduce build time

    alluvial_plot_abs <- alluvial_plot(
      data = tse_long,
      axes = c("Env.broad.scale", "Host.genus", "Diet"),
      abundance_threshold = 2000,
      fill_variable = "Phylum",
      abundance_type = "absolute",
      top_taxa = 5,
      silent = TRUE,
      text_size = 3,
```

```
    legend_ncol = 1,
    custom_colors = DspikeIn::color_palette$cool_MG
  )
  print(alluvial_plot_abs)

}
```

`calculate_list_average_scaling_factors` *Calculate Sample-specific Average Scaling Factors for Multiple Spike-in Groups*

Description

Computes sample-specific scaling factors for multiple groups of spiked species in a `phyloseq` or `TreeSummarizedExperiment` object. Each group can have its own expected spike-in cell count. Scaling factors are calculated per sample and averaged across groups. Missing spike-in observations in a sample will be handled gracefully by averaging available groups.

Usage

```
calculate_list_average_scaling_factors(
  obj,
  spiked_species_list,
  spiked_cells_list,
  merge_method = c("sum", "max")
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`spiked_species_list`	A list of character vectors. Each vector contains taxon names (at species level) for one spike-in group.
`spiked_cells_list`	A numeric vector specifying the expected number of spike-in cells for each group. The order must match `spiked_species_list`.
`merge_method`	Character. Either `"sum"` or `"max"`. Controls how OTUs of each spike-in group are merged.

Details

The function assumes that the taxonomy table has a `Species` column. The output is suitable for downstream absolute quantification pipelines. OTUs belonging to each spike-in group will be merged using the specified `merge_method` ("sum" or "max") to obtain a group-specific spike-in abundance in each sample.

If a sample does not contain any spike-in sequences, a scaling factor of 1 is assigned.

Value

A named numeric vector of sample-specific scaling factors.

Notes

- This function does not modify the input object.
- The returned scaling factors are intended to be used for absolute abundance normalization.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load example phyloseq and TSE objects
  data("physeq", package = "DspikeIn")
  data("tse", package = "DspikeIn")

  # Define spike-in species groups and expected cell counts
  spiked_species_list <- list(
    c("Pseudomonas aeruginosa"),
    c("Escherichia coli"),
    c("Clostridium difficile")
  )
  spiked_cells_list <- c(10000, 20000, 15000)

  # --- Phyloseq example ---
  scaling_phyloseq <- calculate_list_average_scaling_factors(
    physeq,
    spiked_species_list,
    spiked_cells_list,
    merge_method = "sum"
  )
  print(scaling_phyloseq)

  # --- TreeSummarizedExperiment (TSE) example ---
  scaling_tse <- calculate_list_average_scaling_factors(
    tse,
    spiked_species_list,
    spiked_cells_list,
    merge_method = "sum"
  )
  print(scaling_tse)
}
```

---

calculate_spikeIn_factors

Calculate Scaling Factors for Spiked Species in Phyloseq or TSE Object

---

Description

Calculates scaling factors for specified spike-in species or genera in a `phyloseq` or `TreeSummarizedExperiment (TSE)` object. It supports genus/species-level detection, removes spike-ins, merges them, computes scaling factors, and returns a bias-corrected absolute count matrix. The function automatically handles:Species or genus-level spike-in identification, Safe tree and taxonomy synchronization, Volume-based scaling (via the spiked.volume field in metadata), Optional export of intermediate results for traceability (Total_Reads.csv, Spiked_Reads.csv, Scaling_Factors.csv).

Usage

```
calculate_spikeIn_factors(
  obj,
  spiked_cells,
  merged_spiked_species,
  output_path = NULL
)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbiome data.
spiked_cells	A numeric value for the number of spiked cells per unit volume.
merged_spiked_species	A character vector of spiked taxon names (species or genus).
output_path	Optional directory path to save intermediate files (default is NULL).

Value

A list with:

scaling_factors	Named numeric vector of scaling factors per sample.
filtered_obj	Input object with spike-in taxa removed.
spiked_species_reads	Data frame with spike-in reads per sample.
total_reads	Data frame with total reads per sample.
spiked_species_merged	The merged spike-in taxa as a phyloseq object.
tree	Original phylogenetic tree, if available.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE) &&
  requireNamespace("phyloseq", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  spiked_cells <- 1847
 species_name <- spiked_species <- c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")
  merged_spiked_species <- "Tetragenococcus_halophilus"

  # --- Phyloseq Example ---
 spiked_16S_OTU <- phyloseq::subset_samples(physeq_16SOTU, spiked.volume %in% c("2", "1"))
  temp_output_file <- file.path(tempdir(), "merged_physeq_sum.rds")
  output_dir <- file.path(tempdir(), "spikeIn_factors_output")

  merged_physeq_sum <- Pre_processing_species(
    spiked_16S_OTU,
    species_name = merged_spiked_species,
    merge_method = "sum",
    output_file = temp_output_file
  )

  result_physeq <- calculate_spikeIn_factors(
```

```
  merged_physeq_sum,
  spiked_cells = spiked_cells,
  merged_spiked_species = merged_spiked_species,
  output_path = output_dir
)

print(result_physeq$scaling_factors)

if (file.exists(temp_output_file)) unlink(temp_output_file, force = TRUE)
if (dir.exists(output_dir)) unlink(output_dir, recursive = TRUE, force = TRUE)

# --- TSE Example ---
tse_data <- convert_phyloseq_to_tse(physeq_16SOTU)
merged_tse_sum <- Pre_processing_species(
  tse_data,
  species_name = merged_spiked_species,
  merge_method = "sum"
)

result_tse <- calculate_spikeIn_factors(
  merged_tse_sum,
  spiked_cells = spiked_cells,
  merged_spiked_species = merged_spiked_species
)

print(result_tse$scaling_factors)

# --- Final cleanup of any extra leftover RDS files ---
leftover_rds <- list.files(tempdir(), pattern = "merged_physeq.*\\.rds$", full.names = TRUE)
file.remove(leftover_rds[file.exists(leftover_rds)])
}
```

`calculate_spike_percentage` *Calculate Spike Percentage for Specified Taxa in a Phyloseq or TSE Object*

Description

In spike-in based absolute quantitation workflows, the acceptable recovery range of spike-in reads is system dependent, varying with sequencing platform, extraction protocol, and microbial community structure. This function calculates the percentage of reads, categorizes the results as passed or failed, optionally saves the results as DOCX and CSV files.It also visualizes the relationship between observed spike-in abundance and total reads, stratified across user-defined recovery intervals, to empirically determine the optimal range for quality control.

Usage

```
calculate_spike_percentage(
  obj,
  merged_spiked_species = NULL,
  merged_spiked_hashcodes = NULL,
  output_file = NULL,
  passed_range = c(0.1, 11)
)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing the microbial data.
merged_spiked_species	A character vector of spiked taxa names (can be from any taxonomic level).
merged_spiked_hashcodes	A character vector of spiked hashcodes (ASV/OTU IDs) to check in the dataset. Default is NULL.
output_file	A character string specifying the path to save the output files. Default is NULL (no files are written).
passed_range	A numeric vector of length 2 specifying the range of percentages to categorize results as "passed". Default is c(0.1, 11).

Value

A data frame with the following columns:

Sample Sample identifier.

Total_Reads Total number of reads in the sample.

Spiked_Reads Number of reads mapped to the spike-in taxa.

Percentage Percentage of spike-in reads (Spiked_Reads / Total_Reads * 100).

Result Quality control result, either "passed" or "failed", based on the specified range.

See Also

Pre_processing_species, calculate_spike_percentage

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load example phyloseq object
  data("physeq_16SOTU", package = "DspikeIn")

  # ----------- Phyloseq Example -----------
  species_name <- c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")
  merged_spiked_species <- "Tetragenococcus_halophilus"

  # Pre-process the phyloseq object to merge spike-in taxa
  merged_physeq <- Pre_processing_species(
    physeq_16SOTU,
    species_name = species_name,
    merge_method = "sum"
  )

  # Perform spike-in percentage calculation
  output_docx <- file.path(tempdir(), "spike_summary_physeq.docx")
  result_physeq <- calculate_spike_percentage(
    obj = merged_physeq,
    merged_spiked_species = merged_spiked_species,
    output_file = output_docx,
    passed_range = c(0.1, 20)
  )
  print(result_physeq)
```

```
  # ----------- TSE Example -----------
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  merged_tse <- Pre_processing_species(
    tse_16SOTU,
    species_name = species_name,
    merge_method = "sum"
  )

  output_docx_tse <- file.path(tempdir(), "spike_summary_tse.docx")
  result_tse <- calculate_spike_percentage(
    obj = merged_tse,
    merged_spiked_species = merged_spiked_species,
    output_file = output_docx_tse,
    passed_range = c(0.1, 20)
  )
  print(result_tse)

  # Clean up temporary files
  if (file.exists(output_docx)) unlink(output_docx, force = TRUE)
  if (file.exists(output_docx_tse)) unlink(output_docx_tse, force = TRUE)
}
```

`calculate_spike_percentage_list` *Calculate Spike-in Percentage for Specified Taxa*

Description

Computes the percentage of reads attributed to specified spike-in taxa in a `phyloseq` or `TreeSummarizedExperiment` object. The function merges spike-in taxa, computes percentages, classifies samples into "passed" or "failed" based on a user-defined threshold, and optionally exports DOCX and CSV reports.

Usage

```
calculate_spike_percentage_list(
  obj,
  merged_spiked_species,
  output_path = NULL,
  passed_range = c(0.1, 11)
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`merged_spiked_species`	Character vector or list of spike-in species names (at the `Species` level).
`output_path`	Optional. Character string specifying the output path (DOCX). If `NULL` (default) results are not saved.
`passed_range`	Numeric vector of length 2 specifying the accepted percentage range. Default is `c(0.1, 11)`.

Details

The function automatically detects spike-in OTUs based on the `Species` column in the taxonomy table. It works with both `phyloseq` and `TreeSummarizedExperiment` objects and produces QC diagnostics commonly required for spike-in based absolute quantification workflows.

Value

A `data.frame` with:

- Sample
- Total_Reads
- Total_Reads_spiked
- Percentage (of spike-in reads)
- Result ("passed"/"failed")

Notes

- Assumes the taxonomy table contains a column named `Species`.
- Supports both `phyloseq` and `TreeSummarizedExperiment` objects.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load example phyloseq and TSE objects
  data("physeq", package = "DspikeIn")
  data("tse", package = "DspikeIn")

  # Define merged spike-in species list
  spiked_species_list <- c(
    "Pseudomonas aeruginosa",
    "Escherichia coli",
    "Clostridium difficile"
  )

  # Create temporary output paths
  temp_docx <- file.path(tempdir(), "merged_result.docx")
  temp_csv <- sub(".docx", ".csv", temp_docx)

  # --- Phyloseq Example ---
  result_physeq <- calculate_spike_percentage_list(
    obj = physeq,
    merged_spiked_species = spiked_species_list,
    output_path = temp_docx,
    passed_range = c(0.1, 10)
  )
  print(result_physeq)

  # --- TSE Example ---
  result_tse <- calculate_spike_percentage_list(
    obj = tse,
    merged_spiked_species = spiked_species_list,
    output_path = temp_docx,
    passed_range = c(0.1, 10)
  )
```

```
  print(result_tse)

  # Clean up
  if (file.exists(temp_docx)) unlink(temp_docx, force = TRUE)
  if (file.exists(temp_csv)) unlink(temp_csv, force = TRUE)
}
```

`calculate_summary_stats_table` *Calculate Summary Statistics Table*

Description

Computes summary statistics (mean, standard deviation, standard error, and quartiles) for numeric columns of a data frame, and saves the resulting table as a `.csv` file. This utility is designed for quick post-analysis summaries and ensures that all numerical columns are summarized consistently across datasets.

Usage

```
calculate_summary_stats_table(data, output_path = NULL)
```

Arguments

`data`	A data frame containing numeric variables.
`output_path`	Optional. Character string specifying the CSV output path. Default is `"post_eval_summary.csv"`.

Details

This function provides a concise statistical overview of numeric datasets. It computes central tendency, dispersion, and quartile-based spread metrics and writes the results to a `.csv` file suitable for reporting or downstream use.

Value

A data frame containing mean, standard deviation, standard error, first quartile (Q25), median, and third quartile (Q75) for each numeric column.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # --- Phyloseq example ---
  data("physeq", package = "DspikeIn")
  absolute_count <- phyloseq::otu_table(physeq)

  tmp_csv <- file.path(tempdir(), "physeq_summary.csv")
  summary_table_physeq <- calculate_summary_stats_table(
    data = as.data.frame(absolute_count),
    output_path = tmp_csv
  )
  print(summary_table_physeq)
  if (file.exists(tmp_csv)) file.remove(tmp_csv)
```

```
  # --- TSE example ---
  data("tse", package = "DspikeIn")
  tse_counts <- SummarizedExperiment::assay(tse)

  tmp_csv2 <- file.path(tempdir(), "tse_summary.csv")
  summary_table_tse <- calculate_summary_stats_table(
    data = as.data.frame(tse_counts),
    output_path = tmp_csv2
  )
  print(summary_table_tse)
  if (file.exists(tmp_csv2)) file.remove(tmp_csv2)
}
```

`color_palette` *Original, Extended, and Nature-Inspired Color Palette Sequence*

Description

This object provides multiple color palettes for scientific, exploratory, and publication-grade graphics:

- MG: The original palette.
- extended_palette: MG combined with rainbow colors.
- light_MG: A chic pastel palette.
- MG_Awesome: A vibrant and unique 50-color palette for visualization.
- cool_MG: A sophisticated, modern 50-color palette with oceanic, earthy, and high-contrast tones.
- mix_MG: A randomly mixed unique palette.
- vivid_MG: A vivid and bright 50-color palette.
- Mar_palette: A carefully crafted 40-color nature-inspired palette suitable for high-quality publications.

Usage

```
color_palette
```

Format

A list with eight elements:

MG A character vector of the original color codes.

extended_palette A character vector of extended color codes, combining MG with the rainbow palette.

light_MG A character vector of chic pastel color codes.

MG_Awesome A vibrant 50-color palette designed for high-impact visualizations.

cool_MG A sophisticated and modern 50-color palette with oceanic, earthy, and high-contrast tones.

mix_MG A fully combined and randomly mixed unique palette including Mar_palette.

vivid_MG A vivid and bright 50-color palette.

Mar_palette A 40-color palette inspired by nature, suitable for academic and professional publications.

Details

These palettes can be directly used with ggplot2 or other visualization systems supporting manual color scales.

Value

A named list of character vectors, each containing color hex codes for different palettes.

Examples

```
# Example using the Mar_palette
ggplot2::ggplot(mtcars, ggplot2::aes(x = wt, y = mpg, color = factor(cyl))) +
  ggplot2::geom_point(size = 4) +
  ggplot2::scale_color_manual(values = color_palette$Mar_palette) +
  ggplot2::theme_minimal()
```

---

conclusion *Compute Summary Statistics for Spiked Species*

Description

Computes per-sample spike-in summary statistics from a microbiome object (`phyloseq` or `TSE`), generates a spike-in success report using `calculate_spike_percentage()`, and returns both the raw data and a formatted summary table (`flextable`). The function also attempts to extract and retain the phylogenetic tree if present.

Usage

```
conclusion(
  obj,
  merged_spiked_species,
  max_passed_range = 11,
  output_path = "merged_data.docx"
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object containing microbiome data.
`merged_spiked_species`	A character vector of spiked species names.
`max_passed_range`	Numeric, maximum threshold for passing spike percentage.
`output_path`	Character, file path for the `.docx` output from `calculate_spike_percentage()`.

Value

A list containing:

`summary_stats`	A `flextable` summary of the spike statistics.
`full_report`	The full spiked species report as a `data.frame`.
`phy_tree`	The phylogenetic tree (if available).

Examples

```
## ------------------------------
## Example 1: Using phyloseq object
## ------------------------------
library(DspikeIn)
data("physeq_16SOTU", package = "DspikeIn")

# Merge spike-in species
species_name <- c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")
merged_sum <- Pre_processing_species(
  obj = physeq_16SOTU,
  species_name = species_name,
  merge_method = "sum"
)

# Compute summary statistics
output_doc <- file.path(tempdir(), "summary_phyloseq.docx")

results_physeq <- conclusion(
  obj = merged_sum,
  merged_spiked_species = "Tetragenococcus_halophilus",
  max_passed_range = 20,
  output_path = output_doc
)
print(results_physeq$summary_stats)

## ------------------------------------------------
## Example 2: Using TreeSummarizedExperiment object
## ------------------------------------------------
tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)

output_doc_tse <- file.path(tempdir(), "summary_tse.docx")
results_tse <- conclusion(
  obj = tse_16SOTU,
  merged_spiked_species = "Tetragenococcus_halophilus",
  max_passed_range = 20,
  output_path = output_doc_tse
)
print(results_tse$summary_stats)
```

---

convert_categorical_to_factors

Convert Categorical Columns to Factors in Sample Data

Description

Convert Categorical Columns to Factors in Sample Data

Usage

```
convert_categorical_to_factors(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbial data.

Value

A phyloseq object with updated sample data.

Examples

```
data("physeq_16SOTU", package = "DspikeIn")
ps_factor <- convert_categorical_to_factors(physeq_16SOTU)
```

---

convert_phyloseq_to_tse

Convert a phyloseq *Object to a* TreeSummarizedExperiment

Description

Converts a phyloseq object into a TreeSummarizedExperiment (TSE), preserving key biological data components. The function supports retention of:

- OTU abundance matrix (as assay).
- Taxonomic classifications (as rowData).
- Sample metadata (as colData).
- Phylogenetic tree (as rowTree, if available).
- Reference sequences (as referenceSeq, if available).

This allows seamless interoperability between phyloseq and Bioconductor ecosystems.

Usage

```
convert_phyloseq_to_tse(physeq)
```

Arguments

physeq	A valid phyloseq object.

Value

A TreeSummarizedExperiment object with one or more of the following slots:

- assays: OTU count matrix.
- rowData: Taxonomy table.
- colData: Sample metadata.
- rowTree: Phylogenetic tree (if present).
- referenceSeq: Reference sequences (if present).

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Create a small subset for fast execution
  physeq_sub <- phyloseq::prune_taxa(
    phyloseq::taxa_names(physeq_16SOTU)[1:10],
    phyloseq::prune_samples(
      phyloseq::sample_names(physeq_16SOTU)[1:5],
      physeq_16SOTU
    )
  )

  # Example transformation
  tse_sub <- convert_phyloseq_to_tse(physeq_sub)
  tse_sub
}
```

---

convert_to_absolute_counts

Convert Relative ASV/OTU Counts to Absolute Counts

Description

Converts relative ASV counts in a phyloseq or TreeSummarizedExperiment (TSE) object to absolute counts by multiplying ASV counts by provided scaling factors. Ensures phylogenetic tree (rowTree) and reference sequences (refseq) are **retained** if possible.

Usage

```
convert_to_absolute_counts(obj, scaling_factors, output_dir = NULL)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbial data.
scaling_factors	A named numeric vector of scaling factors for each sample.
output_dir	A character string specifying the directory to save the output file (default: NULL).

Value

A list containing:

- absolute_counts: A data frame of absolute counts.
- obj_adj: The modified phyloseq or TreeSummarizedExperiment object.

Note

The output file absolute_counts.csv is written to the specified directory. For CRAN compliance, use tempdir() when saving files inside examples or vignettes.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  spiked_cells <- 1847
 species_name <- spiked_species <- c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")
  merged_spiked_species <- "Tetragenococcus_halophilus"

 spiked_16S_OTU <- phyloseq::subset_samples(physeq_16SOTU, spiked.volume %in% c("2", "1"))
  Spiked_16S_sum_scaled <- Pre_processing_species(
    spiked_16S_OTU,
    species_name,
    merge_method = "sum",
    output_file = file.path(tempdir(), "merged_physeq_sum.rds")
  )

  result <- calculate_spikeIn_factors(
    Spiked_16S_sum_scaled,
    spiked_cells,
    merged_spiked_species
  )
  scaling_factors <- result$scaling_factors

  result_physeq <- convert_to_absolute_counts(
    Spiked_16S_sum_scaled,
    scaling_factors,
    output_dir = tempdir()
  )
  abs_counts_physeq <- result_physeq$absolute_counts
  physeq_adj <- result_physeq$obj_adj

  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  spiked_16S_OTU_TSE <- tse_16SOTU[, tse_16SOTU$spiked.volume %in% c("2", "1")]

  Spiked_16S_sum_tse_scaled <- Pre_processing_species(
    spiked_16S_OTU_TSE,
    species_name,
    merge_method = "sum",
    output_file = file.path(tempdir(), "merged_tse_sum.rds")
  )

  result <- calculate_spikeIn_factors(
    Spiked_16S_sum_tse_scaled,
    spiked_cells,
    merged_spiked_species
  )
  scaling_factors <- result$scaling_factors

  result_tse <- convert_to_absolute_counts(
    spiked_16S_OTU_TSE,
    scaling_factors,
    output_dir = tempdir()
  )
  abs_counts_tse <- result_tse$absolute_counts
  tse_adj <- result_tse$obj_adj
}
```

convert_tse_to_phyloseq *Convert a* TreeSummarizedExperiment *to a* phyloseq *Object*

Description

Converts a TreeSummarizedExperiment (TSE) object into a phyloseq object, preserving key components such as the OTU table, taxonomy, sample metadata, phylogenetic tree, and reference sequences (if present). This enables seamless interoperability between Bioconductor and phyloseq workflows.

Usage

```
convert_tse_to_phyloseq(tse)
```

Arguments

tse	A TreeSummarizedExperiment object, expected to contain: • OTU abundance matrix (assay named "counts"). • Taxonomy data (as rowData). • Sample metadata (as colData). • Optional phylogenetic tree (from rowTree()). • Optional reference sequences (referenceSeq() accessor).

Value

A phyloseq object with all available components (otu_table, tax_table, sample_data, phy_tree, refseq) populated accordingly.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Convert to TSE and back to phyloseq
  tse <- convert_phyloseq_to_tse(physeq_16SOTU)
  phy <- convert_tse_to_phyloseq(tse)
  print(phy)
}
```

create_directory *Create a Directory and Optionally Set as Working Directory*

Description

This function checks if a specified directory exists and creates it if it doesn't. Optionally, it can also set the newly created or existing directory as the working directory.

Usage

```
create_directory(directory_path, set_working_dir = FALSE)
```

Arguments

directory_path	A character string specifying the path of the directory to create.
set_working_dir	A logical value indicating whether to set the directory as the working directory. Default is FALSE.

Value

NULL. The function prints messages indicating whether the directory was created or already exists, and if the working directory was set.

Examples

```
if (interactive()) {
  # Save the current working directory
  old_wd <- getwd()

  # Use a temporary directory for safe example use
  tmp_dir <- file.path(tempdir(), "example_new_dir")

  # Create the directory and set it as the working directory
  create_directory(tmp_dir, set_working_dir = TRUE)

  # Do something inside tmp_dir...

  # Restore the original working directory
  setwd(old_wd)

  # Remove the created directory
  unlink(tmp_dir, recursive = TRUE, force = TRUE)
}
```

---

create_list *Create a List from a Phyloseq or TSE Object*

Description

Create a List from a Phyloseq or TSE Object

Usage

```
create_list(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Value

A list containing the DGE list and updated phyloseq object.

degree_network *Analyze and Visualize a Microbial Network*

Description

This function loads a microbial network from a GraphML file or an igraph object, computes node degree and modularity, assigns node sizes based on degree, and visualizes the network. Edge thickness is determined by weight. It also computes and optionally saves global network metrics.

Usage

```
degree_network(
  graph_path,
  save_metrics = TRUE,
  metrics_path = "Global_Network_Metrics.csv",
  layout_type = "stress"
)
```

Arguments

graph_path	Character or igraph object. Path to the GraphML file or an already loaded igraph object.
save_metrics	Logical. If TRUE, saves the global metrics as a CSV file.
metrics_path	Optional. Path to save the metrics file. Default: "Global_Network_Metrics.csv".
layout_type	Character. Layout algorithm: "stress" (default), "graphopt", "fr", "mds", or "kk".

Value

A list containing:

plot	A ggplot object displaying the network.
metrics	A dataframe with global network metrics.
layout_data	The layout used for plotting.
graph	The annotated igraph object.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
 Complete <- load_graphml(system.file("extdata", "Complete.graphml", package = "DspikeIn"))

  # Save metrics to a temporary file
  temp_metrics <- file.path(tempdir(), "Global_Network_Metrics.csv")

  result <- degree_network(
    graph_path = Complete,
    save_metrics = TRUE,
    metrics_path = temp_metrics
  )

  print(result$metrics)
  print(result$plot)
```

```
  # Clean up temporary file
  unlink(temp_metrics)
}
```

`detect_common_asvs_taxa` *Detect Common ASVs and Taxa from Multiple Phyloseq or TSE Objects*

Description

This function identifies the common Amplicon Sequence Variants (ASVs) and common taxa detected across multiple `phyloseq` or `TreeSummarizedExperiment` (TSE) objects. It extracts ASVs and taxa using the package's accessor functions, finds the common ones, and returns a pruned object containing only the shared features.

Usage

```
detect_common_asvs_taxa(
  obj_list,
  output_common_csv = "common_asvs_taxa.csv",
  output_common_rds = "common_asvs_taxa.rds",
  return_as_df = FALSE
)
```

Arguments

`obj_list`	A list of `phyloseq` or `TreeSummarizedExperiment` objects. All objects in the list must be of the same class.
`output_common_csv`	A character string specifying the path to save the pruned ASV/Taxa object as a CSV file. Default is `"common_asvs_taxa.csv"`. Set to `NULL` to disable saving.
`output_common_rds`	A character string specifying the path to save the pruned ASV/Taxa object as an RDS file. Default is `"common_asvs_taxa.rds"`. Set to `NULL` to disable saving.
`return_as_df`	A logical indicating whether to return the results as a data frame (`TRUE`) or as the original `phyloseq`/`TSE` object (`FALSE`). Default is `FALSE`.

Details

Optionally, the results can be saved to CSV and RDS files.

Value

A pruned `phyloseq` or `TreeSummarizedExperiment` object containing only common ASVs and taxa. If `return_as_df = TRUE`, a data frame with common ASVs/taxa is returned instead.

Examples

```
## Not run:
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Example with phyloseq objects
  common_physeq <- detect_common_asvs_taxa(list(physeq1, physeq2, physeq3))

  # Example with TreeSummarizedExperiment objects
  common_tse <- detect_common_asvs_taxa(list(tse1, tse2, tse3))
}

## End(Not run)
```

ExpData-class *ExpData Virtual Class*

Description

This is a virtual superclass to support S4 object structure in DESeqDataSet extension.

---

extract_neighbors *Extract First and Second Neighbors of a Target Node*

Description

Extracts the first and second neighbors of a given target node in an `igraph` network. Users can provide either an `igraph` object or a GraphML file path (internal or external).

Usage

```
extract_neighbors(graph = NULL, target_node, mode = "all")
```

Arguments

`graph`	Either: • An `igraph` object representing a network, OR • A character string specifying the **file path** to a GraphML file, OR • `NULL` to use `"Complete.graphml"` from `DspikeIn`.
`target_node`	Character. The **name** of the target node.
`mode`	Character. Direction of edges to consider (`"all"`, `"out"`, `"in"`). • `"all"`: Incoming + outgoing edges (default for undirected graphs). • `"out"`: Only outgoing edges. • `"in"`: Only incoming edges.

Value

A list containing:

`first_neighbors`
: Character vector of **first-degree** neighbor names.

`second_neighbors`
: Character vector of **second-degree** neighbor names.

`summary`
: Data frame summarizing the extracted neighbors.

Examples

```
# Load the built-in Complete graph
complete_graph <- load_graphml("Complete.graphml")
result1 <- extract_neighbors(
  graph = complete_graph,
  target_node = "OTU69:Basidiobolus_sp"
)
print(result1$summary)


# Load from an external GraphML file (ensure the file path is correct)
# external_graph <- load_graphml("~/custom_network.graphml")
# extract_neighbors(external_graph, target_node = "SomeNode")
```

`filter_and_split_abundance` *Filter and Split Abundance Data by Threshold*

Description

Filters low-abundance taxa from a `phyloseq` or `TreeSummarizedExperiment` object, splits data into high- and low-abundance groups, and optionally saves results.

Usage

```
filter_and_split_abundance(obj, threshold = 0.01, output_prefix = NULL)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`threshold`	A numeric value indicating the mean abundance threshold for filtering.
`output_prefix`	A character string specifying the filename prefix for saving. If `NULL`, files will not be saved. Default is `NULL`.

Value

A named list with components:

high Subset with taxa having mean abundance > threshold

low Subset with taxa having mean abundance <= threshold

Each returned element contains a single object of the same class as the input (either `phyloseq` or `TreeSummarizedExperiment`).

Examples

```
data("physeq_ITSOTU", package = "DspikeIn")

# Return results without saving
output <- filter_and_split_abundance(physeq_ITSOTU, threshold = 0.05)

# With saving
tse_ITSOTU <- convert_phyloseq_to_tse(physeq_ITSOTU)

output <- filter_and_split_abundance(tse_ITSOTU,
  threshold = 0.05,
  output_prefix = file.path(tempdir(), "abund")
)
```

---

get_long_format_data *Convert a Phyloseq or TSE Object into a Long-Format Data Frame*

Description

Converts a phyloseq or TreeSummarizedExperiment object into a long-format data frame, similar to phyloseq::psmelt(), for compatibility with visualization functions such as alluvial_plot().

Usage

```
get_long_format_data(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Value

A long-format data.frame containing taxonomic, abundance, and sample metadata.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Use a small subset for examples
  physeq_small <- phyloseq::prune_taxa(
    phyloseq::taxa_names(physeq_16SOTU)[1:10],
    phyloseq::prune_samples(
      phyloseq::sample_names(physeq_16SOTU)[1:5],
      physeq_16SOTU
    )
  )

  tse_small <- convert_phyloseq_to_tse(physeq_small)
  melted <- get_long_format_data(tse_small)
  head(melted)
}
```

`get_otu_table` *Extract OTU Tax Metadata from Object*

Description

Retrieves the OTU table from a `phyloseq` or `TreeSummarizedExperiment` object.

Retrieves the OTU table or assay matrix from a `phyloseq` or `TreeSummarizedExperiment` object.

Retrieves the OTU table from a `phyloseq` or `TreeSummarizedExperiment` object.

Retrieves the OTU table from a `phyloseq` or `TreeSummarizedExperiment` object.

Usage

```
get_otu_table(obj)

get_otu_table(obj)

get_otu_table(obj)

get_otu_table(obj)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.

Details

This is a thin wrapper that unifies access to `assay()` (for TreeSummarizedExperiment) and `otu_table()` (for phyloseq) into a single function for format-agnostic use.

Ensures consistent extraction across object classes and automatically transposes the `phyloseq` matrix if taxa are not stored as rows.

Value

A matrix containing OTU count data.

A numeric matrix containing OTU count data.

A matrix containing OTU count data.

A matrix containing OTU count data.

See Also

`otu_table`, `assay`

get_phy_tree *Extract Phylogenetic Tree*

Description

Retrieves the phylogenetic tree.

Usage

```
get_phy_tree(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Details

This function wraps phy_tree() and rowTree() for compatibility with both frameworks.

Value

A phylogenetic tree object.

See Also

phy_tree, rowTree

get_reference_seq *Extract Reference Sequences*

Description

Retrieves the reference sequences (if available) from an object.

Usage

```
get_reference_seq(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Details

Thin wrapper around refseq() or referenceSeq() for unified sequence retrieval.

Value

A DNAStringSet object containing the reference sequences.

See Also

refseq, referenceSeq

get_sample_data *Extract Sample Data*

Description

Retrieves the sample metadata.

Usage

```
get_sample_data(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Details

Thin wrapper over colData() or sample_data() for unified metadata access across formats.

Value

A data frame with sample metadata.

See Also

sample_data, colData, meta

get_sample_sums *Extract Sample Sums from Object*

Description

Retrieves the total sequence counts per sample from a phyloseq or TreeSummarizedExperiment object.

Usage

```
get_sample_sums(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Value

A numeric vector of total counts per sample.

get_tax_table *Extract Taxonomy Table*

Description

Retrieves the taxonomy table from an object.

Usage

```
get_tax_table(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.

Details

This is a thin wrapper around rowData() or tax_table() to ensure compatibility across data structures.

Value

A data frame containing taxonomy annotations.

See Also

tax_table, rowData

gm_mean *Calculate Geometric Mean*

Description

This documentation provides an overview of normalization methods used for microbiome analysis.

- **For Differential Abundance Analysis**: DESeq, TMM, or CSS normalization methods handle compositional biases and library size differences.
- **For Compositional Data**: CLR normalization accounts for compositional structure by transforming the data into log-ratio format.
- **For Simplicity and Ease of Use**: TC, UQ, or Median normalization methods are quick but may not be as robust.

This function calculates the geometric mean of a numeric vector. It removes non-positive and NA values by default.

Usage

```
gm_mean(x, na.rm = TRUE)
```

Arguments

x	A numeric vector.
na.rm	Logical. Should missing values (NAs) be removed? Defaults to TRUE.

Details

Overview of Normalization Methods

Value

Geometric mean of x, or NA if no valid values are present.

Normalization Use Cases

These normalization methods are commonly used in microbiome analysis to ensure fair comparisons across samples.

Examples

```
vec <- c(1, 10, 100, 1000)
gm_mean(vec)
```

---

imbalance_calculate_list_average_scaling_factors

Calculate Per-Sample Scaling Factors for Multiple Spike-in Groups

---

Description

Computes per-sample scaling factors for multiple spike-in taxa (e.g., *Bacillus_spike*, *Flavobacterium_spike*) in either a phyloseq or TreeSummarizedExperiment object. Handles variable spike-in cell counts per sample and supports "sum" or "max" OTU merging methods.

Usage

```
imbalance_calculate_list_average_scaling_factors(
  obj,
  spiked_species_list,
  spiked_cells_list,
  merge_method = c("sum", "max"),
  normalize = TRUE,
  allow_infinite = FALSE,
  verbose = FALSE
)
```

Arguments

obj	A phyloseq::phyloseq or TreeSummarizedExperiment::TreeSummarizedExperiment object.
spiked_species_list	A named list of character vectors giving the spike-in species names (as in tax_table$Species or rowData()).

`spiked_cells_list`
: A named list (same length as `spiked_species_list`) containing scalar or named numeric vectors of expected spike-in cells per sample.

`merge_method`
: `"sum"` (default) or `"max"`. Defines how OTUs within a spike-in group are merged.

`normalize`
: Logical; if `TRUE`, scaling factors are normalized so that their median equals 1. Default = `TRUE`.

`allow_infinite`
: Logical; if `TRUE`, zero spike reads return `Inf` instead of `NA`. Default = `FALSE`.

`verbose`
: Logical; if `TRUE`, prints per-group summaries.

Details

Scaling factors are computed as:

$$ScalingFactor = ExpectedSpikeCells / ObservedSpikeReads$$

For each spike-in group:

1. Identify OTUs matching that spike species via the `Species` column.
2. Merge those OTUs per sample (`sum` or `max`).
3. Divide expected spike cells by observed reads.
4. Average across all spike-in groups to produce one factor per sample.

Uses full matrix preallocation (no incremental vector growth) for Bioconductor compliance. Missing values (zero spike reads) are set to `NA` or `Inf` if `allow_infinite = TRUE`. Samples with all `NA` receive scaling = 1.

Value

Named numeric vector of scaling factors (one per sample).

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  library(phyloseq)

  ## Example dataset
  otu <- matrix(
    c(
      6000, 6200, 5900, 6100,
      4000, 4200, 3900, 4100,
      2000, 1900, 2100, 2050,
      1300, 1250, 1350, 1400,
       500,  800,  900,  700,   # Flavobacterium_spike
       900, 1200, 1100, 1000    # Bacillus_spike
    ),
    nrow = 6, byrow = TRUE,
    dimnames = list(
      c("OTU1", "OTU2", "OTU3", "OTU4",
        "Flavobacterium_spike", "Bacillus_spike"),
      c("S1", "S2", "S3", "S4")
    )
  )

  tax <- data.frame(
```

```
  Kingdom = rep("Bacteria", 6),
  Species = c("OTU1", "OTU2", "OTU3", "OTU4",
              "Flavobacterium_spike", "Bacillus_spike"),
  row.names = rownames(otu)
)

#  Fixed: add a column so sample_data is valid
sam <- data.frame(SampleID = c("S1", "S2", "S3", "S4"),
                  row.names = c("S1", "S2", "S3", "S4"))

ps <- phyloseq(
  otu_table(otu, taxa_are_rows = TRUE),
  tax_table(as.matrix(tax)),
  sample_data(sam)
)

spiked_species_list <- list(
  Flavo = "Flavobacterium_spike",
  Bacillus = "Bacillus_spike"
)

spiked_cells_list <- list(
  Flavo = c(S1 = 1e7, S2 = 3e7, S3 = 6e7, S4 = 2e7),
  Bacillus = c(S1 = 2e7, S2 = 1e7, S3 = 5e7, S4 = 3e7)
)

## Works for both phyloseq and TSE:
factors_phy <- imbalance_calculate_list_average_scaling_factors(
  ps, spiked_species_list, spiked_cells_list, normalize = FALSE
)

tss <- convert_phyloseq_to_tse(ps)
factors_tse <- imbalance_calculate_list_average_scaling_factors(
  tss, spiked_species_list, spiked_cells_list, normalize = FALSE
)

all.equal(factors_phy, factors_tse)
}
```

`label` *Label Taxonomic Ranks by Hashcode*

Description

Labels ASVs/OTUs in a `phyloseq` object using a named vector mapping hashcodes to known taxonomy labels (e.g., spike-in species, genera, or families). This is especially useful for clearly labeling synthetic controls or mock taxa.

If the specified taxonomic rank (e.g., "Genus" or "Family") does not exist in the `tax_table`, it will be added and filled with `NA`, followed by inserting your custom labels.

Usage

```
label(obj, hashcode_label_map, tax_rank = "Species")
```

Arguments

obj	A phyloseq::phyloseq object with a populated tax_table().
hashcode_label_map	A named character vector where names are ASV hashcodes (i.e., row names of the tax table), and values are the taxonomy labels to assign.
tax_rank	A character string specifying the taxonomic rank to label (e.g., "Species", "Genus", "Family"). Default is "Species".

Value

A modified phyloseq object with updated taxonomic labels at the specified rank.

See Also

phyloseq::tax_table()

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  library(phyloseq)

  # Create dummy tax_table with hashcodes
  tax_mat <- matrix(
    data = c(
      "Bacteria", NA,
      "Bacteria", NA,
      "Bacteria", NA,
      "Bacteria", NA
    ),
    nrow = 4,
    dimnames = list(
      c(
        "8ac7ad6e4b6501eb143d97f10bcc2b6d",
        "5a92565231c6df8f58871c0df2d1a12a",
        "8bf5a7b04cb725bc3d2627b971eb03fb",
        "7ae171e44f46ddadcbb53ee6b34a483b"
      ),
      c("Kingdom", "Species")
    )
  )

  ps <- phyloseq::phyloseq(phyloseq::tax_table(tax_mat))

  # Define hash-to-label map
  hash_map <- c(
    "8ac7ad6e4b6501eb143d97f10bcc2b6d" = "MockGenus_A",
    "5a92565231c6df8f58871c0df2d1a12a" = "MockGenus_B"
  )

  # Label genus (automatically adds "Genus" column if missing)
  ps <- label(ps, hash_map, tax_rank = "Genus")

  # Check updated taxonomy
  phyloseq::tax_table(ps)[, "Genus"]
}
```

load_graphml *Load GraphML Without 'id' Conflicts*

Description

Loads a GraphML network from DspikeIn or user path, ensuring all attributes remain intact and correctly assigning original node names.

Usage

```
load_graphml(filename = "herp.spiecsym.network.graphml")
```

Arguments

filename	Name of the GraphML file (default: "herp.spiecsym.network.graphml") or full file path.

Value

An igraph object representing the network.

metadata_full *Metadata for Microbiome Samples*

Description

This dataset contains detailed metadata associated with microbiome sequencing samples. It includes host information, sequencing metrics, environmental classifications, and experimental factors relevant for downstream analyses.

Usage

```
data(metadata_full)
```

Format

A data frame with 312 rows and 46 columns:

sample.id Unique identifier for each sample.

Total_Reads_total Total number of sequencing reads before processing.

Total_Reads_spiked Total number of sequencing reads after spiking.

Percentage Relative abundance percentage of the target taxa.

Result Experimental result classification.

beta.distances Beta diversity distances between samples.

Observed Observed microbial richness in the sample.

Chao1 Chao1 diversity estimator.

ACE Abundance-based Coverage Estimator (ACE).

se.ACE Standard error of ACE.

Shannon Shannon diversity index.

Simpson Simpson diversity index.

InvSimpson Inverse Simpson diversity index.

X16S.biosample 16S rRNA sequencing biosample identifier.

dna.biosample DNA biosample identifier.

data.type General data type classification.

ampliconlibrary.quantification.ng.ul Library quantification (ng/μL).

plate.ID Identifier for the sequencing plate.

well.location Well location on the sequencing plate.

Env.broad.scale Broad-scale environmental classification.

Host.taxon Taxonomic classification of the host.

Host.genus Genus of the host organism.

Host.species Species of the host organism.

Animal.type Classification of the host (e.g., Mammal, Bird).

Animal.ecomode Ecological mode of the host organism.

Clade.Order Taxonomic order of the host.

Family Taxonomic family of the host.

Diet General dietary classification of the host.

Diet.Detailed Detailed dietary classification.

Habitat Habitat description where the sample was collected.

Metamorphosis Indicates whether the species undergoes metamorphosis.

Reproduction Reproductive strategy of the host species.

Ecoregion.III Ecoregion classification (level III).

Ecoregion.IV Ecoregion classification (level IV).

Site Sampling site identifier.

sample.name Human-readable sample name.

biosample.parent Parent biosample identifier.

data.type.1 Secondary data type classification.

ampliconlibrary.quantification.ng.ul.1 Second library quantification (ng/μL).

plate.ID.1 Secondary plate ID (if applicable).

well.location.1 Secondary well location (if applicable).

sample.or.blank Indicates whether the sample is biological or a blank control.

sample.spiked.blank Indicates whether the sample was spiked with a blank.

spiked.volume Volume of spike added to the sample.

swab.presence Indicates whether a swab was used for collection.

MK.spike Molecular spike-in used in the sequencing process.

Value

A data frame containing sample-level metadata for microbiome sequencing analysis.

Examples

```
data(metadata_full)
head(metadata_full)
summary(metadata_full)
```

MG_shapes *Predefined Shape Vector for Plot Styling*

Description

A numeric vector of point shapes (0-25) for use in ggplot2.

Usage

```
MG_shapes
```

Format

An object of class `numeric` of length 26.

Value

A numeric vector of shape codes.

Examples

```
# Example usage of MG_shapes in ggplot2
ggplot2::ggplot(mtcars, ggplot2::aes(x = wt, y = mpg, shape = factor(cyl))) +
  ggplot2::geom_point(size = 4) +
  ggplot2::scale_shape_manual(values = MG_shapes[seq_len(3)]) +
  my_custom_theme()
```

---

my_custom_theme *Custom ggplot2 Theme with Consistent Aesthetics*

Description

Creates a custom ggplot2 theme with consistent styling options, including background color, font size, and axis line formatting.

Usage

```
my_custom_theme(base_size = 12, font_family = "sans", bg_color = "white")
```

Arguments

`base_size`	Numeric. Base font size for text elements (default: 12).
`font_family`	Character. Font family to use for text elements (default: "sans").
`bg_color`	Character. Background color of the plot (default: "white").

Details

This function applies a custom theme to ggplot2 plots. It removes unnecessary grid lines, ensures a clean background, and standardizes text styling.

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
p <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(size = 3) +
  my_custom_theme()
print(p)
```

---

node_level_metrics *Compute and Visualize Node-Level Network Metrics*

Description

Computes various network metrics and generates:

- Two **multi-panel visualizations** (each with 4 subplots).
- A **faceted plot** of standardized (Z-score) metrics across communities.
- A **formatted flextable** summarizing node-level metrics.

Usage

```
node_level_metrics(graph, save_path = NULL)
```

Arguments

graph	An igraph object representing the network.
save_path	Character. A file path (without extension) to save figures and table. Default is NULL (no saving).

Details

The function computes the following node-level metrics:

Metric	Description
Node	Node name (character format)
Degree	Number of edges connected to the node
Strength	Sum of edge weights connected to the node
Closeness	Closeness centrality (normalized, based on shortest paths)
Betweenness	Betweenness centrality (normalized, measures control over network flow)
EigenvectorCentrality	Eigenvector centrality (importance based on connections to influential nodes)
PageRank	PageRank score (importance based on incoming links)
Transitivity	Local clustering coefficient (tendency of a node to form triangles)

Coreness	Node's coreness (from k-core decomposition)
Constraint	Burt's constraint (measures structural holes in a node's ego network)
EffectiveSize	Inverse of constraint (larger values = more non-redundant connections)
Redundancy	Sum of constraint values of a node's alters
Community	Community assignment from Louvain clustering
Efficiency	Global efficiency (average inverse shortest path length)
Local_Efficiency	Local efficiency (subgraph efficiency for a node's neighbors)
Within_Module_Connectivity	Proportion of neighbors in the same community
Among_Module_Connectivity	Proportion of neighbors in different communities

Value

A list containing:

- `metrics`: A data frame with node-level metrics.
- `plot1`: First multi-panel plot (2x2 layout with 4 subplots)
- `plot2`: Second multi-panel plot (2x2 layout with 4 subplots)
- `facet_plot`: A faceted plot showing **Z-score standardized** metrics across communities.

Examples

```
library(igraph)
set.seed(42)
# For external graphml please use full address
# Load internal graphml
Complete <- load_graphml("Complete.graphml")

# Compute node-level metrics
result <- node_level_metrics(Complete)

# View computed metrics
print(result$metrics)

# Show the first 4x4 plot
print(result$plots$plot1)

# Show the second 4x4 plot
print(result$plots$plot2)

# Show facet plot
print(result$facet_plot)

# Print metrics and flextable
print(result$metrics)
print(result$flextable)
```

---

norm.clr *CLR Normalization (Centered Log-Ratio Transformation)*

Description

CLR Normalization (Centered Log-Ratio Transformation)

Usage

```
norm.clr(obj)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.

Value

A list containing the normalized phyloseq object and scaling factors.

---

norm.css *CSS Normalization (Cumulative Sum Scaling)*

Description

CSS Normalization (Cumulative Sum Scaling)

Usage

```
norm.css(obj)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.

Value

A list containing the normalized phyloseq object and scaling factors.

---

norm.DESeq *DESeq Normalization with Pseudocount and Integer Conversion*

Description

DESeq Normalization

Usage

```
norm.DESeq(obj, groups, pseudocount = 1)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.
groups	A string specifying the grouping variable in sample data.
pseudocount	A numeric value added to avoid zeros in the dataset.

Value

A list containing the normalized object (same format as input) and scaling factors.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Example 1: phyloseq input (subset to Animal.type == "Frog")
  physeq_frog <- phyloseq::subset_samples(physeq_16SOTU, Animal.type == "Frog")
  result_DESeq_phy <- norm.DESeq(physeq_frog, groups = "Animal.type", pseudocount = 1)

  # Example 2: TSE input (convert and subset to Animal.type == "Frog")
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  col_meta <- SummarizedExperiment::colData(tse_16SOTU)
  tse_frog <- tse_16SOTU[, which(col_meta$Animal.type == "Frog")]
  result_DESeq_tse <- norm.DESeq(tse_frog, groups = "Animal.type", pseudocount = 1)
}
```

---

norm.med *Median Normalization*

Description

Median Normalization

Usage

```
norm.med(obj, groups)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
groups	A string specifying the grouping variable in sample data.

Value

A list containing the normalized phyloseq object and scaling factors.

---

norm.Poisson *Poisson Normalization and Differential Abundance Function*

Description

Poisson Normalization and Differential Abundance Function

Usage

```
norm.Poisson(obj, group_var = NULL, pseudocount = 1e-06)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
group_var	A string specifying the grouping variable in sample data (if phyloseq object).
pseudocount	A numeric value added to avoid division by zero.

Value

A list containing the normalized data, scaling factor, and differential abundance results.

`norm.QN` *Quantile Normalization (QN) for phyloseq object*

Description

Quantile Normalization (QN) for phyloseq object

Usage

```
norm.QN(obj, filter = FALSE)
```

Arguments

`obj`	A Phyloseq or TreeSummarizedExperiment objects.
`filter`	Logical, whether to filter low counts.

Value

A list containing the normalized phyloseq object and scaling factors.

`norm.rar` *Rarefying*

Description

Rarefying

Usage

```
norm.rar(obj)
```

Arguments

`obj`	A Phyloseq or TreeSummarizedExperiment objects.

Value

A list containing the normalized phyloseq object and scaling factors.

norm.rle *RLE Normalization (Relative Log Expression)*

Description

RLE Normalization (Relative Log Expression)

Usage

```
norm.rle(
  obj,
  locfunc = stats::median,
  type = c("poscounts", "ratio"),
  geo_means = NULL,
  control_genes = NULL
)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
locfunc	A function to compute the location statistic (default is median).
type	A character string specifying the type of normalization ("poscounts" or "ratio").
geo_means	A vector of geometric means for each feature.
control_genes	A vector of control genes.

Value

A list containing the normalized phyloseq object and scaling factors.

norm.TC *TC Normalization (Total Count Scaling)*

Description

TC Normalization (Total Count Scaling)

Usage

```
norm.TC(obj, groups)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
groups	A string specifying the grouping variable in sample data.

Value

A list containing the normalized phyloseq object and scaling factors.

norm.TMM *TMM Normalization (Trimmed Mean of M component)*

Description

TMM Normalization (Trimmed Mean of M component)

Usage

```
norm.TMM(obj, groups)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
groups	A string specifying the grouping variable in sample data.

Value

A list containing the normalized phyloseq object and scaling factors.

norm.tss *TSS Normalization (Total Sum Scaling)*

Description

TSS Normalization (Total Sum Scaling)

Usage

```
norm.tss(obj)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.

Value

A list containing the normalized phyloseq object and scaling factor

norm.UQ *UQ Normalization (Upper Quartile)*

Description

UQ Normalization (Upper Quartile)

Usage

```
norm.UQ(obj, groups)
```

Arguments

obj	A Phyloseq or TreeSummarizedExperiment objects.
groups	A string specifying the grouping variable in sample data.

Value

A list containing the normalized phyloseq object and scaling factors.

normalization_set *Apply the Selected Normalization Method to the Phyloseq and TSE Objects*

Description

Apply the Selected Normalization Method to the Phyloseq and TSE Objects

Usage

```
normalization_set(obj, method, groups = NULL)
```

Arguments

obj	A phyloseq object.
method	A character string specifying the normalization method ("TC", "UQ", "med", "DESeq", "Poisson", "QN", "TMM", "clr", "rar", "css", "tss", "rle").
groups	A column name of group labels from sample data.

Value

A list containing the normalized phyloseq object and scaling factors.

Examples

```
# Example with a phyloseq object
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")
  ps <- physeq_16SOTU
  result_phyloseq <- normalization_set(ps, method = "TC", groups = "Host.species")
  head(result_phyloseq$scaling.factor)
  normed_physeq <- result_phyloseq$dat.normed
}

# Example with a TreeSummarizedExperiment (TSE) object
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  result_tse <- normalization_set(tse_16SOTU, method = "clr")
  head(result_tse$scaling.factor)
  normed_tse <- result_tse$dat.normed
}

# For a full comparison of all normalization methods, see the vignette:
# vignette("DspikeIn-with-Phyloseq", package = "DspikeIn")
# vignette("DspikeIn-with-TSE", package = "DspikeIn")
```

---

perform_and_visualize_DA

Perform and Visualize Differential Abundance Analysis with edgeR or DESeq2

Description

Performs differential abundance analysis using edgeR or DESeq2, and visualizes results with a volcano plot, a log-fold change bar plot, and a relative abundance bar plot for significantly enriched taxa. Supports comparisons across one or more variables and allows global FDR correction across multiple contrasts.

Usage

```
perform_and_visualize_DA(
  obj,
  method,
  group_var = NULL,
  contrast,
  pseudocount = 1,
  significance_level = 0.05,
  output_csv_path = NULL,
  target_glom = "Genus",
  palette = c("#FFEB3B", "#073B4C"),
  global_fdr = TRUE
)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.
method	A string: either "edgeR" or "DESeq2".
group_var	A string specifying the grouping variable in the sample metadata. Not required if contrast is a named list for multiple variables.
contrast	One of the following: • A character vector of two levels to compare (e.g., c("Control", "Treated")). • A list of such character vectors for multiple contrasts within one grouping variable. • A named list of such lists (e.g., list(Treatment = list(c("A", "B"), c("A", "C")), Genotype = list(...))).
pseudocount	A numeric value used to replace zero or negative counts (default = 1).
significance_level	A numeric value for the FDR threshold to determine significance (default = 0.05).
output_csv_path	Optional path to save results as CSV files. For multiple contrasts, each is saved separately.
target_glom	A string specifying the taxonomic rank to aggregate taxa (default = "Genus").
palette	A vector of colors for plotting significant and non-significant points (default = c("#FFEB3B", "#073B4C")).
global_fdr	Logical. If TRUE, applies FDR correction across all contrasts (default = FALSE).

Details

For edgeR, the standard error of log-fold change (lfcSE) is estimated using the formula: lfcSE = logFC / sqrt(LR), based on the likelihood ratio test statistic.

When providing multiple contrasts, FDR correction can be applied globally using the global_fdr = TRUE option.

Value

A list of result objects, or a single result if a single contrast is given. Each result includes:

results	A data.frame of differential abundance results with taxonomic annotations.
obj_significant	A filtered phyloseq or TreeSummarizedExperiment object with significant taxa.
plot	A ggplot2 volcano plot.
bar_plot	A ggplot2 bar plot of log fold changes and standard errors.
bar_abundance_plot	A ggplot2 bar plot showing relative abundance of significant taxa across groups.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Salamander samples belonging to two diet types
  ps_sal <- phyloseq::subset_samples(
    physeq_16SOTU,
    Animal.type == "Salamander" &
      Diet %in% c("Insectivore", "Carnivore")
  )

  # Remove taxa with zero counts
  ps_sal <- phyloseq::prune_taxa(
    phyloseq::taxa_sums(ps_sal) > 0,
    ps_sal
  )

  # Differential abundance test at Genus level
  da_res <- perform_and_visualize_DA(
    obj = ps_sal,
    method = "DESeq2",
    group_var = "Diet",
    significance_level = 0.05,
    contrast = c("Insectivore", "Carnivore"),
    target_glom = "Genus"
  )

  # Visualize results
  if (!is.null(da_res$plot)) print(da_res$plot)
  head(da_res$results)

  # Example: multiple contrasts (optional demonstration)
  contrast_list <- list(c("Insectivore", "Carnivore"))
  da_multi <- perform_and_visualize_DA(
    obj = ps_sal,
    method = "DESeq2",
    group_var = "Diet",
    significance_level = 0.01,
    contrast = contrast_list,
    target_glom = "Genus",
    global_fdr = TRUE
  )

  if (!is.null(da_multi[[1]]$bar_plot))
    print(da_multi[[1]]$bar_plot)
}
```

---

physeq *Example Phyloseq Object with Tree and Reference Sequences*

Description

A general-purpose phyloseq object for microbiome method development and testing. This synthetic dataset includes OTU abundances, taxonomic classifications, sample metadata, a phylogenetic tree, and reference sequences.

Usage

```
data(physeq)
```

Format

A phyloseq object with:

otu_table OTU abundance matrix (ASVs samples).

tax_table Taxonomic classification of ASVs.

sample_data Sample-level metadata.

phy_tree A rooted phylogenetic tree of ASVs.

refseq DNA reference sequences corresponding to each ASV.

Value

A phyloseq object with full microbiome data components.

Source

Synthetic data generated for benchmarking and demonstration.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  data(physeq, package = "DspikeIn")
  physeq
  phyloseq::sample_names(physeq)
  phyloseq::taxa_names(physeq)
  phyloseq::phy_tree(physeq)
  phyloseq::refseq(physeq)
}
```

---

physeq_16SOTU *Example Phyloseq Object for 16S OTUs*

Description

This dataset contains an example phyloseq object representing 16S rRNA gene amplicon sequencing data. It includes taxonomic assignments, abundance counts, sample metadata, and phylogenetic information. This object is intended for demonstration and testing of microbiome workflows.

Usage

```
data(physeq_16SOTU)
```

Format

A phyloseq object with:

otu_table Operational Taxonomic Unit (OTU) abundance matrix.

tax_table Taxonomic classification of OTUs.

sample_data Metadata associated with the samples.

phy_tree Phylogenetic tree relating OTUs (if available).

Value

A phyloseq object containing 16S OTU data with taxonomy, sample metadata, and phylogeny.

Source

Internal dataset for microbiome analysis.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  data(physeq_16SOTU)
  physeq_16SOTU
  summary(physeq_16SOTU)
  phyloseq::sample_names(physeq_16SOTU)
  phyloseq::taxa_names(physeq_16SOTU)
}
```

---

physeq_ITSOTU *Example Phyloseq Object for ITS OTUs*

Description

This dataset contains an example phyloseq object representing Internal Transcribed Spacer (ITS) amplicon sequencing data. It includes taxonomic annotations, OTU abundance counts, and associated sample metadata, suitable for downstream analysis of fungal communities.

Usage

```
data(physeq_ITSOTU)
```

Format

A phyloseq object with:

otu_table Operational Taxonomic Unit (OTU) abundance matrix.

tax_table Taxonomic classification of OTUs.

sample_data Metadata associated with the samples.

phy_tree Phylogenetic tree relating OTUs (if available).

Value

A phyloseq object containing ITS OTU data with taxonomy, sample metadata, and phylogeny.

Source

Internal dataset for microbiome analysis.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  data(physeq_ITSOTU)
  physeq_ITSOTU
  summary(physeq_ITSOTU)
  phyloseq::sample_data(physeq_ITSOTU)
  phyloseq::taxa_names(physeq_ITSOTU)
}
```

---

plotbar_abundance *Taxa Bar Plot Without Aggregation (Relative or Absolute Abundance)*

Description

Create a bar plot of relative or absolute abundances of microbial taxa **without using taxonomic glomming** (`tax_glom`). Works on raw or normalized data.

Usage

```
plotbar_abundance(
  physeq,
  tax_level = "Genus",
  normalize = TRUE,
  treatment_variable = "Host.taxon",
  abundance_type = "relative",
  x_angle = 25,
  fill_variable = tax_level,
  facet_variable = NULL,
  palette = DspikeIn::color_palette$mix_MG,
  legend_size = 11,
  legend_columns = 1,
  x_scale = "free",
  xlab = NULL
)
```

Arguments

`physeq`	A `phyloseq` or `TreeSummarizedExperiment` object.
`tax_level`	A character string indicating the taxonomic rank (e.g., `"Genus"`).
`normalize`	Logical; if `TRUE`, transforms to relative abundance. Default: `TRUE`.
`treatment_variable`	Character; column name in sample metadata for x-axis grouping.
`abundance_type`	Character; either `"relative"` or `"absolute"`. Default: `"relative"`.
`x_angle`	Numeric; angle of x-axis tick labels. Default: `25`.
`fill_variable`	Character; variable to fill bars by. Default: same as `tax_level`.

facet_variable	Optional; column name for faceting. Default: NULL.
palette	A named vector of colors to use for fill_variable.
legend_size	Numeric; legend text size. Default: 11.
legend_columns	Integer; number of legend columns. Default: 1.
x_scale	Character; either "free" or "fixed" for facet x-axis scale. Default: "free".
xlab	Optional; override x-axis label. If NULL, x label is hidden.

Value

A ggplot2 object containing a bar plot of taxa abundance.

Examples

```
## Not run:
# Load required package
if (requireNamespace("phyloseq", quietly = TRUE)) {
  # Load example data
  data("physeq_ITSOTU", package = "DspikeIn")

  # Subset: Eurycea salamanders from Blue Ridge, exclude unwanted genera
  Des <- physeq_ITSOTU |>
    phyloseq::subset_taxa(Genus != "Dekkera") |>
    phyloseq::subset_samples(Clade.Order == "Caudate") |>
    phyloseq::subset_samples(Host.genus == "Eurycea") |>
    phyloseq::subset_samples(Ecoregion.III == "Blue Ridge")

  # Clean taxa: remove NAs or blanks in Phylum, filter low-abundance
  Des_filtered <- phyloseq::subset_taxa(Des, !is.na(Phylum) & Phylum != "")
  Des_ps <- phyloseq::prune_taxa(phyloseq::taxa_sums(Des_filtered) > 99, Des_filtered)

  # Plot taxa abundance with full control
  plotbar_abundance(
    physeq = Des_ps,
    normalize = TRUE,
    treatment_variable = "Diet",
    abundance_type = "absolute",
    x_angle = 0,
    fill_variable = "Phylum",
    palette = DspikeIn::color_palette$mix_MG,
    legend_size = 10,
    legend_columns = 1,
    x_scale = "free",
    xlab = NULL
  )
}

## End(Not run)
```

`plot_core_microbiome_custom` *Plot Core Microbiome Prevalence Heatmap (Phyloseq & TSE Compatible)*

Description

This function generates a prevalence heatmap of the core microbiome at a specified taxonomic rank. It allows users to pass custom detection thresholds, prevalence thresholds, and a minimum prevalence filter, and it provides an option to order taxa either in ascending or descending abundance. The plot displays a heatmap showing detection thresholds at different prevalence levels.

Usage

```
plot_core_microbiome_custom(
  obj,
  taxrank = "Genus",
  select_taxa = NULL,
 detections = list(prevalences = seq(0.03, 1, 0.01), thresholds = 10^seq(log10(0.03),
    log10(1), length = 10), min_prevalence = 0.2, taxa_order = "descending"),
  output_core_csv = NULL,
  output_core_rds = NULL
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` (TSE) object containing microbiome data.
`taxrank`	A character string specifying the taxonomic rank to glom taxa. Default is "Genus".
`select_taxa`	A character vector of taxa to select. Default is `NULL`, meaning no specific taxa are selected.
`detections`	A list with the following elements: • `prevalences`: A numeric vector specifying the prevalence thresholds for plotting. Default is `seq(0.03, 1, 0.01)`. • `thresholds`: A numeric vector specifying the detection thresholds for plotting. Default is `10^seq(log10(3e-2), log10(1), length = 10)`. • `min_prevalence`: A numeric value specifying the minimum prevalence threshold for core microbiome. Default is 0.2. • `taxa_order`: A character string indicating whether to order taxa by "ascending" or "descending" abundance. Default is "descending".
`output_core_csv`	Path to save the core microbiome subset as a CSV file. Default is `NULL`. To avoid writing to the working directory during examples, use `file.path(tempdir(), "core_microbiome.csv")`.
`output_core_rds`	Path to save the core microbiome subset as an RDS file. Default is `NULL`. Use `file.path(tempdir(), "core_microbiome.rds")` to write to a temporary directory during examples or checks.

Value

A ggplot2 object representing the core microbiome prevalence heatmap.

Source

Uses microbiome::plot_core() for core heatmap visualization

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Subset to only Frog samples (biologically meaningful)
  physeq_frog <- phyloseq::subset_samples(
    physeq_16SOTU,
    Animal.type == "Frog"
  )

  # Remove taxa with zero total abundance
  physeq_frog <- phyloseq::prune_taxa(
    phyloseq::taxa_sums(physeq_frog) > 0,
    physeq_frog
  )

  # Relaxed thresholds to ensure visible core taxa
  custom_detections <- list(
    prevalences    = seq(0.03, 1, 0.01),
    thresholds     = 10^seq(log10(0.03), log10(1), length = 10),
    min_prevalence = 0.3,
    taxa_order     = "ascending"
  )

  # Temporary output paths (required for Bioconductor examples)
  core_csv <- file.path(tempdir(), "core_microbiome_frog.csv")
  core_rds <- file.path(tempdir(), "core_microbiome_frog.rds")

  # Generate the core microbiome prevalence plot
  plot_result <- plot_core_microbiome_custom(
    obj = physeq_frog,
    detections = custom_detections,
    taxrank = "Genus",
    output_core_csv = core_csv,
    output_core_rds = core_rds
  )

  print(plot_result)
}
```

---

plot_spikein_tree_diagnostic

Spike-in Tree Diagnostic Plot

Description

Diagnostic visualization of spike-in taxa (ASVs/OTUs) on a phylogenetic tree. Works with both `phyloseq` and `TreeSummarizedExperiment` objects (auto-converts internally).

Visual elements:

- Tip labels (OTU/ASV names)
- Branch length annotations
- Prevalence (star size on tip)
- Log10(mean abundance) (bar ring)
- Sample metadata (discrete tile at tip)

Usage

```
plot_spikein_tree_diagnostic(
  obj,
  metadata_var,
  output_prefix = "spikein_diag",
  layout = "circular",
  save_plot = FALSE,
  width = 10,
  height = 10
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object filtered to spike-in taxa.
`metadata_var`	Character. Sample metadata variable to use as tile color.
`output_prefix`	Character. Filename prefix for saved plot. Default = "spikein_diag".
`layout`	Character. Tree layout. Default = "circular".
`save_plot`	Logical. Save figure if TRUE.
`width`	Numeric. Width of the output plot in inches. Default = 10.
`height`	Numeric. Height of the output plot in inches. Default = 10.

Value

Invisibly returns the ggplot object.

Examples

```
## Not run:
if (
  requireNamespace("DspikeIn", quietly = TRUE) &&
    requireNamespace("phyloseq", quietly = TRUE) &&
    requireNamespace("TreeSummarizedExperiment", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggtree", quietly = TRUE) &&
    requireNamespace("ggtreeExtra", quietly = TRUE) &&
    requireNamespace("ggstar", quietly = TRUE) &&
    requireNamespace("ggnewscale", quietly = TRUE)
) {
  # ----------- Phyloseq Example -----------
```

```
  data("physeq_16SOTU", package = "DspikeIn")
  spikein_ps <- phyloseq::subset_taxa(physeq_16SOTU, Genus == "Tetragenococcus")

  plot_spikein_tree_diagnostic(
    obj = spikein_ps,
    metadata_var = "Animal.type",
    save_plot = FALSE
  )

  # ----------- TSE Example -----------
  tse_spikein <- convert_phyloseq_to_tse(spikein_ps)

  plot_spikein_tree_diagnostic(
    obj = tse_spikein,
    metadata_var = "Animal.type",
    save_plot = FALSE
  )
}

## End(Not run)
```

`Pre_processing_hashcodes` *Pre-process phyloseq or TSE object based on hashcodes*

Description

Subsets, merges, and saves taxa based on hashcodes and a specified merge method ("sum" or "max"). This function pre-processes a `phyloseq` or `TreeSummarizedExperiment` (TSE) object by subsetting, merging, and saving taxa based on provided hashcodes. It retains taxonomic information and creates intermediate datasets for further downstream analysis.

Usage

```
Pre_processing_hashcodes(
  obj,
  hashcodes,
  merge_method = c("sum", "max"),
  output_prefix = "merged_physeq"
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`hashcodes`	A character vector of taxon hashcodes (OTU row names).
`merge_method`	The method to merge taxa: "sum" or "max".
`output_prefix`	A prefix for the output file names.

Value

A processed phyloseq or TSE object.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Subset to Tetragenococcus species
  tetragenococcus_physeq <- phyloseq::subset_taxa(
    physeq_16SOTU,
    Species %in% c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")
  )

  # Extract OTU IDs (hashcodes) for phyloseq object
  hashcodes_physeq <- rownames(phyloseq::otu_table(tetragenococcus_physeq))

  # Remove previous output file if exists
  if (file.exists("merged_physeq_processed.rds")) {
    file.remove("merged_physeq_processed.rds")
  }

  # Run merging with "sum" method for phyloseq
  processed_sum <- Pre_processing_hashcodes(
    physeq_16SOTU,
    hashcodes = hashcodes_physeq,
    merge_method = "sum"
  )

  # Convert to TreeSummarizedExperiment (TSE)
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  tetragenococcus_TSE <- convert_phyloseq_to_tse(tetragenococcus_physeq)

  # Extract hashcodes for TSE
  hashcodes_tse <- rownames(tetragenococcus_TSE)

  # Run merging with "max" method for TSE
  processed_max <- Pre_processing_hashcodes(
    tse_16SOTU,
    hashcodes = hashcodes_tse,
    merge_method = "max"
  )

  # Final cleanup of written file
  file.remove("merged_physeq_processed.rds")
}
```

`Pre_processing_species` *Pre-process taxa in a phyloseq or TSE object by merging ASVs/OTUs*

Description

Merges ASVs/OTUs while ensuring that the phylogenetic tree and reference sequences remain intact. The provided taxonomic name(s) will be searched across **all taxonomic levels** (e.g., Kingdom, Phylum, ..Genus, Species). If tree or refseq become mismatched, they are pruned or removed safely.

Usage

```
Pre_processing_species(
  obj,
  species_name,
  merge_method = c("sum", "max"),
  output_file = NULL
)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object.
species_name	A character vector of **exact** taxonomic names to merge (matched across all taxonomy levels).
merge_method	Method used to merge counts: "sum" (default) or "max".
output_file	Optional file path to save the processed object (e.g., file.path(tempdir(), "output.rds")).

Value

A processed phyloseq or TreeSummarizedExperiment object with merged ASVs/OTUs.

Examples

```
library(DspikeIn)
data("physeq_16SOTU", package = "DspikeIn")

species_name <- c("Tetragenococcus_halophilus", "Tetragenococcus_sp.")

# Merge species in phyloseq format
merged_sum <- Pre_processing_species(
  physeq_16SOTU,
  species_name,
  merge_method = "sum"
)

# Convert phyloseq to TSE format
tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)

# Merge species in TSE format and write to tempdir
output_rds <- file.path(tempdir(), "merged_TSE_sum.rds")

merged_TSE_sum <- Pre_processing_species(
  tse_16SOTU,
  species_name,
  merge_method = "sum",
  output_file = output_rds
)
```

`Pre_processing_species_list` *Preprocess and Merge Spike-in Species in a Phyloseq or TSE Object*

Description

Merges ASVs belonging to user-defined spike-in species by summing or selecting maximum counts, while preserving all available metadata (taxonomy, sample data, phylogenetic tree, and reference sequences). This function works for both `phyloseq` and `TreeSummarizedExperiment` objects.

Usage

```
Pre_processing_species_list(
  obj,
  spiked_species,
  merge_method = c("sum", "max"),
  output_file = NULL
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`spiked_species`	Character vector of species names to be processed (matched against the `Species` column in taxonomy).
`merge_method`	Either `"sum"` (default) to sum counts across ASVs or `"max"` to retain only the most abundant ASV.
`output_file`	Optional. File path to save the merged object as an `.rds` file.

Value

A merged object of the same class as the input (`phyloseq` or `TreeSummarizedExperiment`).

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq", package = "DspikeIn")
  data("tse", package = "DspikeIn")

 spiked_species <- c("Pseudomonas aeruginosa", "Escherichia coli", "Clostridium difficile")

  # Merge in phyloseq object
  merged_physeq <- Pre_processing_species_list(
    physeq,
    spiked_species = spiked_species,
    merge_method = "sum"
  )

  # Merge in TreeSummarizedExperiment object
  merged_tse <- Pre_processing_species_list(
    tse,
    spiked_species = spiked_species,
    merge_method = "sum"
```

```
  )
}
```

---

proportion_adj *Proportionally Adjust Abundance*

Description

This function normalizes the abundance data in a phyloseq or TreeSummarizedExperiment object by adjusting each sample's counts based on a total value, typically the maximum total sequence count across all samples. The adjusted counts are then rounded to the nearest integer.

Usage

```
proportion_adj(obj, output_file = "proportion_adjusted.rds")
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbiome data.
output_file	A character string specifying the output file name for the adjusted object. If NULL, the object is not saved. Default is "proportion_adjusted.rds".

Details

This function extracts the OTU table (or assay in TSE), normalizes it based on the sample sums, and updates the original object while maintaining its structure.

Value

A modified object of the same class (phyloseq or TreeSummarizedExperiment) with proportionally adjusted and rounded abundance data.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load phyloseq object
  data("physeq_16SOTU", package = "DspikeIn")

  normalized_physeq <- proportion_adj(
    physeq_16SOTU,
    output_file = file.path(tempdir(), "proportion_adjusted_physeq.rds")
  )
  print(normalized_physeq)

  # Convert to TSE and apply
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  normalized_tse <- proportion_adj(
    tse_16SOTU,
    output_file = file.path(tempdir(), "proportion_adjusted_tse.rds")
  )
  print(normalized_tse)
}
```

quadrant_plot *Generate Custom Quadrant Plots for Node Metrics*

Description

This function visualizes relationships between **any two node metrics** in a quadrant plot. Quadrant labels dynamically adjust based on the selected X and Y axis metrics.

Usage

```
quadrant_plot(
  metrics,
  x_metric = "Degree",
  y_metric = "Redundancy",
  x_threshold = NULL,
  y_threshold = NULL,
  top_quantile = 0.95,
  point_size = 3
)
```

Arguments

metrics	A data.frame containing computed node metrics.
x_metric	Character. Column name to use for the x-axis. Default is "Degree".
y_metric	Character. Column name to use for the y-axis. Default is "Redundancy".
x_threshold	Numeric. X-axis threshold for quadrant separation. Default is median(x_metric).
y_threshold	Numeric. Y-axis threshold for quadrant separation. Default is median(y_metric).
top_quantile	Numeric. Quantile threshold (0-1) to highlight top nodes. Default is 0.95.
point_size	Numeric. Size of the points. Default is 3.

Value

A ggplot object representing the customized quadrant plot.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  g <- load_graphml("Complete.graphml")

  # Compute node-level metrics
  result <- node_level_metrics(g)
  metrics <- result$metrics

  # Generate a quadrant plot using Degree and Efficiency
  plot <- quadrant_plot(metrics, x_metric = "Degree", y_metric = "Efficiency")
  print(plot)
}
```

RandomForest_selected *Select Important ASVs/OTUs Using Random Forest*

Description

This function selects the most important Amplicon Sequence Variants (ASVs) or Operational Taxonomic Units (OTUs) based on a Random Forest model. If a TreeSummarizedExperiment (TSE) is provided, it is first converted to phyloseq. The function allows filtering and pruning of taxa before selecting the most important features. Optionally, the selected ASVs/OTUs can be saved as a CSV file.

Usage

```
RandomForest_selected(
  physeq,
  response_var,
  minlib = 5000,
  prunescale = 1e-05,
  ntree = 100,
  n_top_predictors = 100,
  output_csv = NULL,
  na_vars = NULL
)
```

Arguments

physeq	A phyloseq or TreeSummarizedExperiment (TSE) object containing microbiome data.
response_var	A character string specifying the response variable from the sample metadata.
minlib	A numeric value specifying the minimum library size for filtering low-abundance taxa. Default is 15000.
prunescale	A numeric value specifying the relative abundance threshold for pruning rare OTUs. Default is 0.0001.
ntree	An integer specifying the number of trees to grow in the Random Forest model. Default is 100.
n_top_predictors	An integer specifying the number of top ASVs/OTUs to select based on feature importance. Default is 50.
output_csv	An optional character string specifying the output CSV file name. If NULL, no file is saved. Default is NULL.
na_vars	A character vector specifying metadata variables to check for missing values (NA). If NULL, only response_var is checked.

Value

Returns a pruned phyloseq or TreeSummarizedExperiment (TSE) object containing only the selected ASVs/OTUs. If the input is TSE, the output is converted back to TSE.

Source

Based on public API usage of randomForest and phyloseq packages.

See Also

randomForest, prune_taxa

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Perform Random Forest feature selection
  rf_physeq <- RandomForest_selected(
    physeq_16SOTU,
    prunescale = 0.00001,
    minlib = 5000,
    ntree = 30,
    n_top_predictors = 30,
    response_var = "Host.genus",
    na_vars = c("Habitat", "Ecoregion.III", "Host.genus", "Diet")
  )
  # Less aggressive pruning (retain rare taxa)
  rf_physeq_relaxed <- RandomForest_selected(
    physeq_16SOTU,
    response_var = "Host.genus",
    minlib = 5000,
    prunescale = 0.00001,
    na_vars = c("Habitat", "Ecoregion.III", "Host.genus", "Diet")
  )

  rf_physeq_strict <- RandomForest_selected(
    physeq_16SOTU,
    response_var = "Host.genus",
    minlib = 20000,
    prunescale = 0.0002,
    ntree = 200,
    n_top_predictors = 30,
    na_vars = c("Habitat", "Ecoregion.III", "Host.genus", "Diet")
  )

  # Load TreeSummarizedExperiment (TSE) object
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)

  # Perform Random Forest feature selection on TSE object
  rf_tse <- RandomForest_selected(
    tse_16SOTU,
    response_var = "Host.genus",
    na_vars = c("Habitat", "Ecoregion.III", "Host.genus", "Diet")
  )
}
```

randomsubsample_Trimmed_evenDepth
Subsampling to an Equal Sequencing Depth

Description

Performs subsampling to an equal sequencing depth, determined by the sample with the lowest sequencing depth after excluding very low abundant taxa. It rounds down the result to the nearest integer (floor). Note: some samples may be lost in this process.

Usage

```
randomsubsample_Trimmed_evenDepth(
  obj,
  smalltrim = 0.001,
  replace = TRUE,
  output_file = NULL
)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbiome data.
smalltrim	A numeric value specifying the trimming percentage to exclude very low abundant taxa. Default is 0.001.
replace	A logical value indicating whether to sample with replacement. Default is TRUE.
output_file	A character string specifying the output file name for the subsampled object. Default is NULL, meaning the object will not be saved.

Value

A rarefied phyloseq or TreeSummarizedExperiment object with adjusted sequencing depths.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_ITSOTU", package = "DspikeIn")
  tse_ITSOTU <- convert_phyloseq_to_tse(physeq_ITSOTU)
  rarefied <- randomsubsample_Trimmed_evenDepth(tse_ITSOTU, smalltrim = 0.001)
  print(rarefied)
}
```

`random_subsample_WithReductionFactor` *Random Subsampling with Reduction Factor*

Description

Performs random subsampling on the OTU table of a `phyloseq` or `TreeSummarizedExperiment` (TSE) object by dividing each ASV count by a specified reduction factor and rounding down to the nearest whole number. Optionally, the result can be saved to disk as an `.rds` file.

Usage

```
random_subsample_WithReductionFactor(
  obj,
  reduction_factor = 3,
  output_file = NULL
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`reduction_factor`	A numeric value ≥ 1 to reduce counts. Default is 3.
`output_file`	Optional. A character string specifying the `.rds` file path to save the result. If `NULL`, no file will be saved. Default is `NULL`.

Value

A subsampled object of the same class as the input (`phyloseq` or `TreeSummarizedExperiment`).

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE) &&
  requireNamespace("phyloseq", quietly = TRUE) &&
  requireNamespace("TreeSummarizedExperiment", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")
  red <- random_subsample_WithReductionFactor(physeq_16SOTU, reduction_factor = 10)
  summary_stats <- summ_phyloseq_sampleID(red)
  print(summary_stats)

  tse <- convert_phyloseq_to_tse(physeq_16SOTU)
  red_tse <- random_subsample_WithReductionFactor(tse, reduction_factor = 10)
  summary_stats <- summ_phyloseq_sampleID(red_tse)
  print(summary_stats)
}
```

`regression_plot` *Create a Regression Plot with Faceting by Range*

Description

This function generates a customizable scatter plot with a linear regression line, statistical equation, and facets based on a specified range variable. The x and y variables are transformed using a natural log transformation (`log1p`) to handle zero values.

Usage

```
regression_plot(
  data,
  x_var,
  y_var,
  custom_range = c(0.1, 15, 30, 50, 75, 100),
  formula = y ~ x,
  plot_title = NULL
)
```

Arguments

`data`	A data frame containing the variables to plot.
`x_var`	A string specifying the name of the x-axis variable.
`y_var`	A string specifying the name of the y-axis variable.
`custom_range`	A numeric vector for defining custom ranges for the 'Percentage' column (default: c(0.1, 15, 30, 50, 75, 100)).
`formula`	A formula for the regression equation (default: y ~ x).
`plot_title`	A string specifying the title of the plot (default: NULL, no title will be shown if not provided).

Value

A ggplot2 object.

See Also

`stat_regline_equation`, `stat_cor`, `facet_wrap`

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("metadata_full", package = "DspikeIn")

  plot_object <- regression_plot(
    data = metadata_full,
    x_var = "Observed",
    y_var = "Total_Reads_spiked",
    custom_range = c(0.1, 15, 30, 50, 75, 100)
  )
```

```
  # Print the plot output
  print(plot_object)
}
```

`relativized_filtered_taxa` *Filter Taxa from a Phyloseq or TSE Object Based on Custom Thresholds*

Description

This function filters taxa from a phyloseq or TreeSummarizedExperiment (TSE) object based on custom thresholds for percentage of samples, mean abundance, count, and relative abundance.

Usage

```
relativized_filtered_taxa(
  obj,
  threshold_percentage = 0.5,
  threshold_mean_abundance = 0.001,
  threshold_count = 10,
  threshold_relative_abundance = NULL
)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object.
`threshold_percentage`	A numeric value specifying the minimum percentage of samples in which a taxon must be present to be retained. Default is 0.5.
`threshold_mean_abundance`	A numeric value specifying the minimum mean abundance of a taxon to be retained. Default is 0.001.
`threshold_count`	A numeric value specifying the minimum count of a taxon in a sample to be considered present. Default is 10.
`threshold_relative_abundance`	A numeric value specifying the minimum relative abundance of a taxon to be retained. Default is NULL.

Value

A filtered phyloseq or TSE object containing only the taxa that meet the specified thresholds.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Apply relative filtering on taxa
  FT <- relativized_filtered_taxa(
    physeq_16SOTU,
```

```
    threshold_percentage = 0.001,
    threshold_mean_abundance = 1,
    threshold_count = 5,
    threshold_relative_abundance = 0.001
  )
}
```

`remove_zero_negative_count_samples` *Remove Samples with Zero, Negative Counts, or NA Values and Add Pseudocount*

Description

Remove Samples with Zero, Negative Counts, or NA Values and Add Pseudocount

Usage

```
remove_zero_negative_count_samples(obj, pseudocount = 1e-06)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object containing microbial data.
`pseudocount`	A numeric value to add to avoid zero counts.

Value

A phyloseq object with filtered and adjusted OTU table.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  library(DspikeIn)
  data("physeq_16SOTU", package = "DspikeIn")

  # Remove samples with zero/negative/NA counts and add pseudocount
  cleaned_ps <- remove_zero_negative_count_samples(
    physeq_16SOTU,
    pseudocount = 1e-6
  )
}
```

`ridge_plot_it` *Generate Ridge Plots for Taxonomic Abundance Distribution*

Description

This function processes microbiome data (`phyloseq` or `TreeSummarizedExperiment`), rarefies the dataset, performs proportion transformation, and generates ridge plots to visualize the distribution of relative abundances at a specified taxonomic rank.

Usage

```
ridge_plot_it(obj, taxrank = "Genus", rarefaction_depth = NULL, top_n = 10)
```

Arguments

`obj`	A `phyloseq` or `TreeSummarizedExperiment` object containing taxonomic and abundance data.
`taxrank`	A character string specifying the taxonomic rank for glomming and plotting. Default is "Genus".
`rarefaction_depth`	A numeric value specifying the rarefaction depth. If NULL, it is set to 90\ of the minimum sample sum (for `phyloseq`). Default is NULL.
`top_n`	An integer specifying the number of top taxa to include in the plot. Default is 10.

Details

This function:

- Rarefies the dataset (if `phyloseq`) to normalize sample depth.
- Extracts abundance and taxonomic data using `get_otu_table()` and `get_tax_table()`.
- Aggregates abundance data at the specified taxonomic rank.
- Selects the top `n` taxa by total abundance.
- Generates a ridge plot to visualize abundance distributions.

Value

A `ggplot2` object representing the ridge plot of the distribution of relative abundances.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load phyloseq object
  data("physeq_16SOTU", package = "DspikeIn")
  ridge_physeq <- ridge_plot_it(physeq_16SOTU, taxrank = "Family", top_n = 10)


  # convert phyloseq object to TSE
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  ridge_tse <- ridge_plot_it(tse_16SOTU, taxrank = "Family", top_n = 10)
}
```

set_nf *Set Normalization Factors in the Sample Data of the Phyloseq Object*

Description

Set Normalization Factors in the Sample Data of the Phyloseq Object

Usage

```
set_nf(obj, scaling.factor)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbial data.
scaling.factor	A vector of normalization factors.

Value

A phyloseq object with updated sample data.

Examples

```
if (requireNamespace("phyloseq", quietly = TRUE)) {
  data("physeq_ITSOTU", package = "DspikeIn")

  # Create normalization factors (e.g., all ones)
  nf <- rep(1, phyloseq::nsamples(physeq_ITSOTU))

  # Apply normalization factors
  physeq_ITSOTU <- DspikeIn::set_nf(physeq_ITSOTU, scaling.factor = nf)

  # Check the updated sample data
  head(phyloseq::sample_data(physeq_ITSOTU)$norm_factors)
}
```

---

simulate_network_robustness *Simulate Network Robustness under Node Removal*

Description

Evaluates the robustness of a network by iteratively removing nodes and measuring the size of the largest connected component at each step.

Usage

```
simulate_network_robustness(
  graph,
  steps = 10,
  removal_strategy = "random",
  plot_results = TRUE
)
```

Arguments

graph	An igraph object representing the network.
steps	Integer. Number of nodes to remove. Default is 10.
removal_strategy	Character. Node removal strategy, one of: • "random" (default): Removes nodes randomly. • "degree": Removes the highest-degree nodes first. • "betweenness": Removes the highest-betweenness nodes first.
plot_results	Logical. If TRUE, generates and **returns** a plot of the robustness curve. Default is TRUE.

Details

Users can specify a node removal strategy from the following options:

- **"random"**: Removes nodes randomly.
- **"degree"**: Removes the node with the highest degree first.
- **"betweenness"**: Removes the node with the highest betweenness centrality first.

Important: For the "betweenness" strategy, all edge weights **must be positive**.

Value

A **list** containing:

- results: A data.frame with the step number and the relative size of the largest connected component.
- plot: A ggplot2 object (returned if plot_results = TRUE).

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  Complete <- load_graphml("Complete.graphml")

  # Simulate robustness by removing 200 highest-degree nodes
  robustness_degree <- simulate_network_robustness(
    graph = Complete,
    steps = 200,
    removal_strategy = "degree"
  )

  # Simulate robustness with random node removal
  robustness_random <- simulate_network_robustness(
    graph = Complete,
    steps = 200,
    removal_strategy = "random"
  )

  # Simulate robustness with betweenness-based node removal
  robustness_betweenness <- simulate_network_robustness(
    graph = Complete,
    steps = 200,
    removal_strategy = "betweenness"
```

```
  )

  # Print robustness plots
  print(robustness_degree$plot)
  print(robustness_random$plot)
  print(robustness_betweenness$plot)
}
```

---

summ_ASV_OTUID *Summarize ASV Data Based on ASV_ID*

Description

This function generates summary statistics (mean, median, standard deviation, standard error, and quantiles) for each ASV (Amplicon Sequence Variant) in a phyloseq or TreeSummarizedExperiment object.

Usage

```
summ_ASV_OTUID(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing ASV abundance data.

Details

This function extracts the OTU table (or assay in TSE), computes per-ASV statistics, and returns a tidy summary data frame.

Value

A data frame containing summary statistics for each ASV.

Examples

```
# Example with a phyloseq object
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_ITSOTU", package = "DspikeIn")
  summary_physeq <- summ_ASV_OTUID(physeq_ITSOTU)

  # Example with a TreeSummarizedExperiment object
  tse_ITSOTU <- convert_phyloseq_to_tse(physeq_ITSOTU)
  summary_tse <- summ_ASV_OTUID(tse_ITSOTU)
}
```

summ_count_phyloseq *Summary Statistics of a Phyloseq or TSE Object*

Description

Computes overall summary statistics (mean, median, standard deviation, standard error, and quantiles) for the OTU table in a phyloseq or TreeSummarizedExperiment (TSE) object.

Usage

```
summ_count_phyloseq(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing taxonomic and abundance data.

Value

A data frame with overall summary statistics.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Summarize counts for the phyloseq object
  summary_stats_physeq <- summ_count_phyloseq(physeq_16SOTU)

  # Convert phyloseq object to a TreeSummarizedExperiment (TSE)
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)

  # Summarize counts for the TSE object
  summary_stats_tse <- summ_count_phyloseq(tse_16SOTU)
}
```

summ_phyloseq_sampleID *Generate Summary Statistics for Each Sample*

Description

Calculates summary statistics (mean, median, standard deviation, standard error, and quartiles) for each sample in a phyloseq or TreeSummarizedExperiment object.

Usage

```
summ_phyloseq_sampleID(obj)
```

Arguments

obj	A phyloseq or TreeSummarizedExperiment object containing microbiome data.

Value

A data frame containing summary statistics per sample, with columns:

- Sample_ID
- Mean
- Median
- Standard Deviation (SD)
- Standard Error (SE)
- Q25 (25th percentile)
- Q50 (Median)
- Q75 (75th percentile)

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")

  # Summarize the phyloseq object
  summary_stats_physeq <- summ_phyloseq_sampleID(physeq_16SOTU)
  print(summary_stats_physeq)

  # Convert to TreeSummarizedExperiment
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  summary_stats_tse <- summ_phyloseq_sampleID(tse_16SOTU)
  print(summary_stats_tse)
}
```

---

taxa_barplot *Generate a Taxa Barplot with Relative or Absolute Abundance*

Description

This function creates a bar plot of relative or absolute abundances of the top n taxa (OTUs/ASVs) at a specified taxonomic rank. Non-top taxa are aggregated into an "Others" category, which is always displayed at the top for clarity. The function supports normalization, faceting, and customization of plot aesthetics. The plot is built using ggplot2, and the taxa are ordered with "Others" appearing first.

Usage

```
taxa_barplot(
  physeq,
  target_glom = "Genus",
  custom_tax_names = NULL,
  normalize = TRUE,
```

```
  treatment_variable = "Treatment",
  abundance_type = "relative",
  x_angle = 25,
  fill_variable = target_glom,
  facet_variable = NULL,
  top_n_taxa = 20,
  palette = color_palette$MG,
  legend_size = 11,
  legend_columns = 1,
  x_scale = "free",
  xlab = NULL
)
```

Arguments

physeq	A phyloseq or TreeSummarizedExperiment object containing microbiome data.
target_glom	A character string specifying the taxonomic rank to plot (e.g., "Genus").
custom_tax_names	A character vector specifying custom taxonomic names for the levels. Default is NULL.
normalize	A logical value indicating whether to normalize sample counts to relative abundances. Default is TRUE.
treatment_variable	A character string specifying the treatment variable for the x-axis (e.g., "Treatment"). Default is "Treatment".
abundance_type	A character string specifying whether to plot "relative" or "absolute" abundance. Default is "relative".
x_angle	A numeric value specifying the angle of x-axis text labels. Default is 25.
fill_variable	A character string specifying the variable to use for filling bar colors. Default is the same as target_glom.
facet_variable	A character string specifying the variable to use for faceting the plot. Default is NULL (no faceting).
top_n_taxa	A numeric value specifying the number of top taxa to include in the plot. Default is 20.
palette	A character vector of color codes or a function generating such a palette. Default is color_palette$MG.
legend_size	A numeric value specifying the size of the legend text. Default is 11.
legend_columns	A numeric value specifying the number of columns for the legend. Default is 1.
x_scale	A character string specifying the x-axis scale in facets. Options are "free" or "fixed". Default is "free".
xlab	A character string specifying the x-axis label. Default is NULL (no label).

Value

A list containing the following components:

barplot A ggplot2 object representing the taxa barplot.

taxa_data A phyloseq object containing the top taxa and aggregated "Others" taxa.

Source

Built on public functions from phyloseq and ggplot2 for data transformation and plotting.

Examples

```
# Example 1: Relative abundance barplot (subset to Frog samples for speed)
data("physeq_16SOTU", package = "DspikeIn")

# Subset to only 'Frog' samples
physeq_frog <- phyloseq::subset_samples(physeq_16SOTU, Animal.type == "Frog")

# Remove taxa with zero abundance
physeq_frog <- phyloseq::prune_taxa(phyloseq::taxa_sums(physeq_frog) > 0, physeq_frog)

# Plot relative abundance for the top 5 genera
bp_rel <- taxa_barplot(
  physeq = physeq_frog,
  target_glom = "Genus",
  fill_variable = "Family",
  treatment_variable = "Diet",
  abundance_type = "relative",
  top_n_taxa = 5,
  legend_size = 9,
  x_scale = "fixed",
  legend_columns = 1,
  x_angle = 15,
  palette = DspikeIn::color_palette$MG
)
print(bp_rel$barplot)


# Example 2: Absolute abundance barplot (ITS data converted to TSE; Frog subset)
data("physeq_ITSOTU", package = "DspikeIn")

# Convert the phyloseq object to TreeSummarizedExperiment
tse_ITSOTU <- convert_phyloseq_to_tse(physeq_ITSOTU)
tse_frog <- tse_ITSOTU[, SummarizedExperiment::colData(tse_ITSOTU)$Animal.type == "Frog"]
tse_frog <- tse_frog[rowSums(SummarizedExperiment::assay(tse_frog)) > 0, ]

# Plot absolute abundance for top 5 Families
bp_abs <- taxa_barplot(
  physeq = tse_frog,
  target_glom = "Family",
  treatment_variable = "Diet",
  fill_variable = "Family",
  abundance_type = "absolute",
  top_n_taxa = 5,
  x_angle = 15,
  legend_size = 9,
  legend_columns = 1,
  x_scale = "fixed",
  palette = DspikeIn::color_palette$cool_MG
)
print(bp_abs$barplot)
```

`tidy_phyloseq_tse` *Tidy a Phyloseq or TreeSummarizedExperiment Object*

Description

Cleans and standardizes a microbiome dataset, supporting both `phyloseq` and `TreeSummarizedExperiment`. Performs:

- Standardization of taxonomic ranks (if available)
- Removal of leading/trailing whitespace in taxa names
- Filtering out zero-count taxa
- Exclusion of "Chloroplast" and "Mitochondria" classifications (if applicable)

Cleans and standardizes a microbiome object by:

- Removing taxonomic prefixes (e.g. `k__`, `p__`)
- Trimming whitespace
- Renaming `Domain` to `Kingdom` (if present)
- Removing taxa with "Chloroplast" or "Mitochondria" in **any** rank
- Filtering out taxa with zero total abundance
- Ensuring robust handling of both phyloseq and TreeSummarizedExperiment formats

Usage

```
tidy_phyloseq_tse(obj)

tidy_phyloseq_tse(obj)
```

Arguments

`obj`	A `phyloseq::phyloseq` or `TreeSummarizedExperiment::TreeSummarizedExperiment` object.

Details

Tidy a Phyloseq Object and Remove Zero/Negative Count Samples

This function standardizes taxonomic ranks, removes unnecessary whitespace, and filters unwanted classifications, ensuring consistency for downstream analysis.

Value

A cleaned and tidied object of the same class.

A cleaned and filtered object of the same class with updated taxonomy.

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  data("physeq_16SOTU", package = "DspikeIn")
  tidy_physeq <- tidy_phyloseq_tse(physeq_16SOTU)
}

if (requireNamespace("DspikeIn", quietly = TRUE)) {
  # Load example phyloseq object
  data("physeq_16SOTU", package = "DspikeIn")

  # ----------- Tidy phyloseq object -----------
  tidy_physeq <- tidy_phyloseq_tse(physeq_16SOTU)

  # ----------- Tidy TSE object -----------
  tse_16SOTU <- convert_phyloseq_to_tse(physeq_16SOTU)
  tidy_tse <- tidy_phyloseq_tse(tse_16SOTU)
}
```

---

tse *Example TreeSummarizedExperiment (TSE) Object with Tree and Reference Sequences*

Description

This dataset contains synthetic microbiome data stored as a `TreeSummarizedExperiment` (TSE) object. The TSE structure includes abundance data, sample metadata, taxonomic annotations, a phylogenetic tree linking ASVs, and reference sequences. It is suitable for benchmarking preprocessing workflows involving taxonomic merging or filtering.

Usage

```
data(tse)
```

Format

A `TreeSummarizedExperiment` object with:

assays Matrix of observed counts or abundances (ASVs samples).

rowData Taxonomic annotations per ASV.

colData Sample-level metadata.

rowTree A phylogenetic tree object representing ASV relationships.

referenceSeq A `DNAStringSet` of DNA sequences matching ASVs.

Value

A `TreeSummarizedExperiment` object with full microbiome context (abundance, taxonomy, tree, and reference sequences).

Source

Simulated data generated for reproducibility in microbiome pipelines.

See Also

TreeSummarizedExperiment::TreeSummarizedExperiment()

Examples

```
if (requireNamespace("TreeSummarizedExperiment", quietly = TRUE)) {
  data(tse, package = "DspikeIn")
  tse
  SummarizedExperiment::assay(tse)
  SummarizedExperiment::colData(tse)
  SummarizedExperiment::rowData(tse)
  TreeSummarizedExperiment::rowTree(tse)
  TreeSummarizedExperiment::referenceSeq(tse)
}
```

---

validate_spikein_clade

Validate Spike-In Clade Consistency with NJ Tree and Bootstrap

---

Description

Validates whether sample spike-in sequences form a monophyletic clade with known reference spike-in(s) using a Neighbor-Joining (NJ) tree with Jukes-Cantor correction and bootstrap support.

This function produces:

- A bootstrap-annotated NJ tree
- A boxplot comparing branch lengths
- A histogram of patristic distances

If `output_prefix` is provided, outputs are saved. Otherwise, they are shown interactively and recorded in-memory.

Usage

```
validate_spikein_clade(
  reference_fasta,
  sample_fasta,
  bootstrap = 100,
  output_prefix = NULL
)
```

Arguments

`reference_fasta`	Character. Path to FASTA file of reference spike-ins.
`sample_fasta`	Character. Path to FASTA file of sample spike-ins.
`bootstrap`	Integer. Number of bootstrap replicates (default = 100).
`output_prefix`	Character or NULL. File prefix for saving output.

Value

A list with:

tree NJ tree (class phylo)

monophyly TRUE if sample spike-ins form a clade

clade_bootstrap Bootstrap support percentage

branch_stats Branch length summary

patristic_distances Patristic distance matrix

tree_plot Tree plot object

branch_boxplot Boxplot object

patristic_histogram Histogram object

summary_text Text summary

alignment Multiple sequence alignment (MSA)

aln_phydat Alignment converted to phyDat

distance_matrix JC69 distance matrix

Examples

```
ref_fasta <- system.file("extdata", "Ref.fasta", package = "DspikeIn")
sample_fasta <- system.file("extdata", "Sample.fasta", package = "DspikeIn")
result <- validate_spikein_clade(ref_fasta, sample_fasta)
```

---

weight_Network *Analyze and Visualize a Microbial Network*

Description

Loads a microbial network from the **DspikeIn** package (Complete.graphml, NoHubs.graphml, or NoBasid.graphml) or from a user-provided **GraphML** file. Computes network metrics, assigns modularity-based node colors (cool_MG from **DspikeIn**), and visualizes the network using ggraph. Optionally saves computed **global network metrics** as a CSV file.

Usage

```
weight_Network(graph_path = NULL, save_metrics = TRUE)
```

Arguments

graph_path	Character. The **GraphML file path**. Default (NULL) loads "Complete.graphml" from **DspikeIn**. Valid internal options: "Complete.graphml", "NoHubs.graphml", "NoBasid.graphml". If an **external path** is provided, the function loads that instead.
save_metrics	Logical. If TRUE, saves the global metrics as a CSV file (in the working directory or specified by metrics_path).

Value

A list containing:

`plot`	A `ggplot2` object displaying the network.
`metrics`	A `data.frame` with global network metrics.
`graph`	The annotated `igraph` object.

See Also

`cluster_fast_greedy`, `ggraph`, `load_graphml`

Examples

```
if (requireNamespace("DspikeIn", quietly = TRUE)) {
  Complete <- load_graphml("Complete.graphml")

  # Run network weighting function on the loaded dataset
  # Load a specific GraphML dataset from DspikeIn
  result <- weight_Network(graph_path = "Complete.graphml")

  # Load a custom GraphML file from user directory,
  # for external graphml please use **full address**
  print(result$plot)

  # View network metrics
  result$metrics
  # Optional: Clean up generated metrics file if saved
  unlink("Global_Network_Metrics.csv")
}
```

Index