Orchestrating Microbiome Analysis

Authors: Leo Lahti [aut], Tuomas Borman [aut, cre], Henrik Eckermann [ctb], Sudarshan Shetty [aut], Felix GM Ernst [aut]
Version: 0.98.12 **Modified:** 2023-02-19 **Compiled:** 2023-03-30
Environment: R version 4.2.1 (2022-06-23), Bioconductor 3.15

https://github.com/microbiome/OMA

Contents

Welcome

You are reading the online book, **Orchestrating Microbiome Analysis with R and Bioconductor** (Lahti et al., 2021b), where we walk through common strategies and workflows in microbiome data science.

The book shows through concrete examples how you can take advantage of the latest developments in R/Bioconductor for the manipulation, analysis, and reproducible reporting of hierarchical and heterogeneous microbiome profiling data sets. The book was borne out of necessity, while updating microbiome analysis tools to work with Bioconductor classes that provide support for multi-modal data collections. Many of these techniques are generic and widely applicable in other contexts as well.

This work has been heavily influenced by other similar resources, in particular the Orchestrating Single-Cell Analysis with Bioconductor (Amezquita et al., 2020a), phyloseq tutorials (Callahan et al., 2016b) and microbiome tutorials (Shetty and Lahti, 2019). This book extends these resources to teach the grammar of Bioconductor workflows in the context of microbiome data science. As such, it supports the adoption of general skills in the analysis of large, hierarchical, and multi-modal data collections. We focus on microbiome analysis tools, including entirely new, partially updated as well as previously established methods.

This online resource and its associated ecosystem of microbiome data science tools are a result of a community-driven development process, and welcoming new contributors. Several individuals have contributed methods, workflows and improvements as acknowledged in the Introduction. You can find more information on how to find us online and join the developer community through the project homepage at microbiome.github.io. This online resource has been written in RMarkdown with the bookdown R package. The material is **free to use** with the Creative Commons Attribution-NonCommercial 3.0 License.

Part I

Introduction

Chapter 1

Introduction

This work - **Orchestrating Microbiome Analysis with R and Bioconductor** (Lahti et al., 2021b) - contributes novel methods and educational resources for microbiome data science. It aims to teach the grammar of Bioconductor workflows in the context of microbiome data science. We show through concrete examples how to use the latest developments and data analytical strategies in R/Bioconductor for the manipulation, analysis, and reproducible reporting of hierarchical, heterogeneous, and multi-modal microbiome profiling data. The data science methodology is tightly integrated with the broader R/Bioconductor ecosystem that focuses on the development of of high-quality open research software for life sciences (Gentleman et al. (2004), Huber et al. (2015)). The support for modularity and interoperability is a key to efficient resource sharing and collaborative development both within and across research fields. The central data infrastructure, the `SummarizedExperiment` data container and its derivatives, have already been widely adopted in microbiome research, single cell sequencing, and in other fields, allowing a rapid adoption and extensions of emerging data science techniques across application domains.

We assume that the reader is already familiar with R programming. For references and tips on introductory material for R and Bioconductor, see Chapter 16. This online resource and its associated ecosystem of microbiome data science tools are a result of a community-driven development process, and welcoming new users and contributors. You can find more information on how to find us online and join the developer community through the project homepage at microbiome.github.io.

The book is organized into three parts. We start by introducing the material and link to further resources for learning R and Bioconductor. We describe the key data infrastructure, the `TreeSummarizedExperiment` class that provides a container for microbiome data, and how to get started by loading microbiome data set in the context of this new framework. The second section, *Focus Topics*, covers the common steps in microbiome data analysis, beginning with the

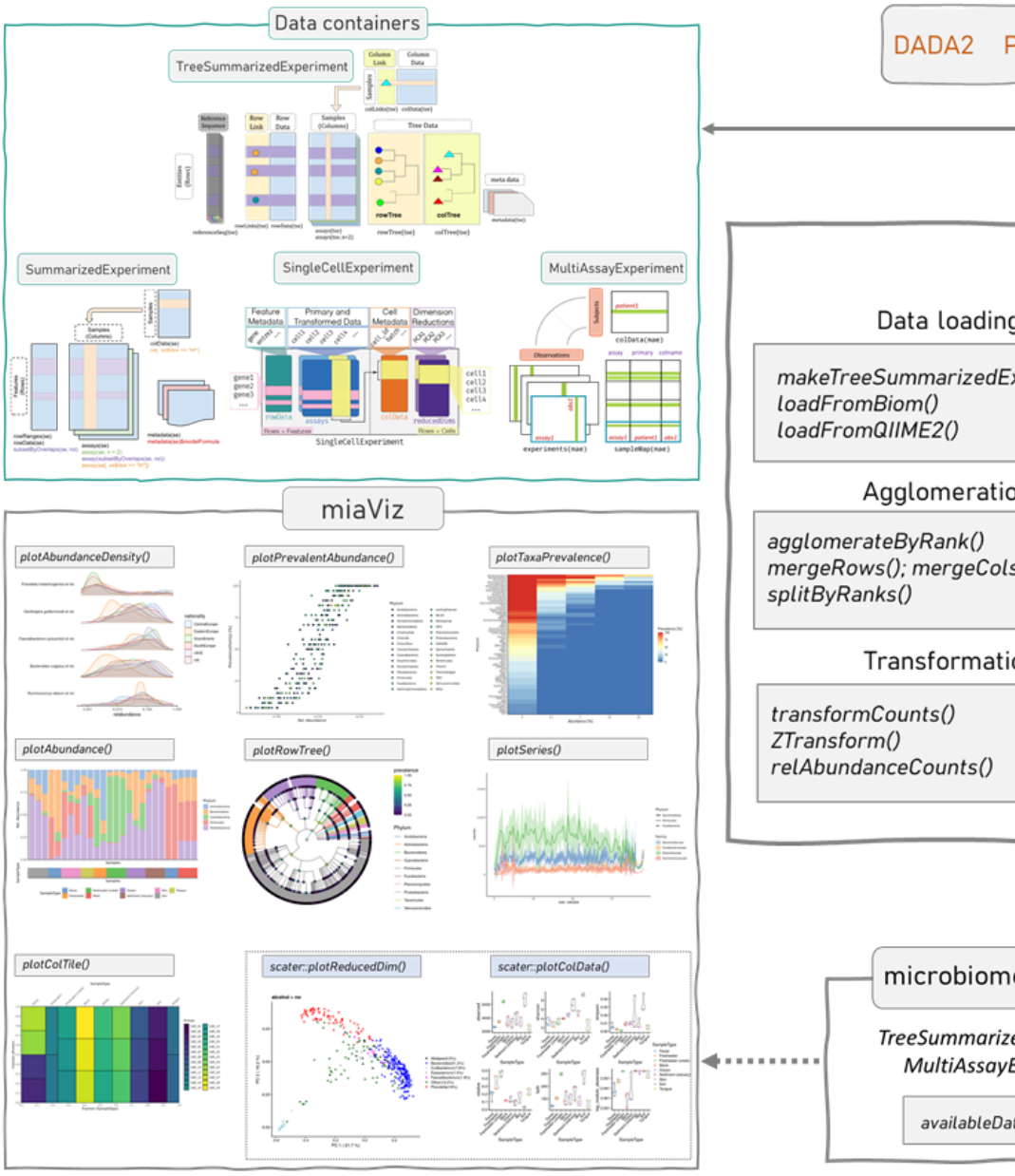
most common steps and progressing to more specialized methods in subsequent sections. Third, *Workflows*, provides case studies for the various datasets used throughout the book. Finally, *Appendix*, links to further resources and acknowledgments.

Chapter 2

Microbiome Data

2.1 Data science framework

The building blocks of the framework are **data container** (SummarizedExperiment and its derivatives), **packages** from various developers using the TreeSE container, open **demonstration data sets**, in a separate chapter 2.4, and **online tutorials** including this online book as well as the various package vignettes and other material.



TreeSE image source: https://f1000research.com/slides/9-1464
SE image source: https://www.bioconductor.org/packages/devel/bioc/vignettes/SummarizedExperiment/inst/doc/SummarizedExper
MAE image source: https://waldronlab.io/MultiAssayWorkshop/articles/Ramos_MultiAssayExperiment.html
SCE s image source: http://bioconductor.org/books/3.13/OSCA.intro/the-singlecellexperiment-class.html

2.2 Data containers

`SummarizedExperiment` (`SE`) (Morgan et al., 2020) is a generic and highly optimized container for complex data structures. It has become a common choice for analysing various types of biomedical profiling data, such as RNAseq, ChIp-Seq, microarrays, flow cytometry, proteomics, and single-cell sequencing.

[`TreeSummarizedExperiment`] (`TreeSE`) (Huang, 2020) was developed as an extension to incorporate hierarchical information (such as phylogenetic trees and sample hierarchies) and reference sequences.

[`MultiAssayExperiment`] (`MAE`) (Ramos et al., 2017) provides an organized way to bind several different data structures together in a single object. For example, we can bind microbiome data (in `TreeSE` format) with metabolomic profiling data (in `SE`) format, with shared sample metadata. This is convenient and robust for instance in subsetting and other data manipulation tasks. Microbiome data can be part of multiomics experiments and analysis strategies and we want to outline the understanding in which we think the packages explained and used in this book relate to these experiment layouts using the `TreeSummarizedExperiment` and classes beyond.

This section provides an introductions to these data containers. In microbiome data science, these containers link taxonomic abundance tables with rich side information on the features and samples. Taxonomic abundance data can be obtained by 16S rRNA amplicon or metagenomic sequencing, phylogenetic microarrays, or by other means. Many microbiome experiments include multiple versions and types of data generated independently or derived from each other through transformation or agglomeration. We start by providing recommendations on how to represent different varieties of multi-table data within the `TreeSummarizedExperiment` class.

The options and recommendations are summarized in Table 2.1.

2.2.1 Assay data

The original count-based taxonomic abundance tables may have different transformations, such as logarithmic, Centered Log-Ratio (CLR), or relative abundance. These are typically stored in ***assays***.

```
library(mia)
data(GlobalPatterns, package="mia")
tse <- GlobalPatterns
assays(tse)
```

```
## List of length 1
## names(1): counts
```

The `assays` slot contains the experimental data as count matrices. Multiple matrices can be stored the result of `assays` is actually a list of matrices.

```
assays(tse)
```

```
## List of length 1
## names(1): counts
```

Individual assays can be accessed via `assay`

```
assay(tse, "counts")[1:5,1:7]
```

```
##        CL3 CC1 SV1 M31Fcsw M11Fcsw M31Plmr M11Plmr
## 549322   0   0   0       0       0       0       0
## 522457   0   0   0       0       0       0       0
## 951      0   0   0       0       0       0       1
## 244423   0   0   0       0       0       0       0
## 586076   0   0   0       0       0       0       0
```

To illustrate the use of multiple assays, the relative abundance data can be calcualted and stored along the original count data using `relAbundanceCounts`.

```
tse <- relAbundanceCounts(tse)
assays(tse)
```

```
## List of length 2
## names(2): counts relabundance
```

Now there are two assays available in the `tse` object, `counts` and `relabundance`.

```
assay(tse, "relabundance")[1:5,1:7]
```

```
##        CL3 CC1 SV1 M31Fcsw M11Fcsw M31Plmr   M11Plmr
## 549322   0   0   0       0       0       0 0.000e+00
## 522457   0   0   0       0       0       0 0.000e+00
## 951      0   0   0       0       0       0 2.305e-06
## 244423   0   0   0       0       0       0 0.000e+00
## 586076   0   0   0       0       0       0 0.000e+00
```

Here the dimension of the count data remains unchanged. This is in fact a requirement for any `SummarizedExperiment` object.

2.2.2 colData

`colData` contains data on the samples.

```
colData(tse)
```

```
## DataFrame with 26 rows and 7 columns
##         X.SampleID   Primer Final_Barcode Barcode_truncated_plus_T
##           <factor> <factor>      <factor>                  <factor>
## CL3        CL3      ILBC_01        AACGCA                    TGCGTT
## CC1        CC1      ILBC_02        AACTCG                    CGAGTT
## SV1        SV1      ILBC_03        AACTGT                    ACAGTT
## M31Fcsw    M31Fcsw  ILBC_04        AAGAGA                    TCTCTT
## M11Fcsw    M11Fcsw  ILBC_05        AAGCTG                    CAGCTT
## ...            ...      ...           ...                       ...
## TS28         TS28   ILBC_25        ACCAGA                    TCTGGT
## TS29         TS29   ILBC_26        ACCAGC                    GCTGGT
## Even1        Even1  ILBC_27        ACCGCA                    TGCGGT
## Even2        Even2  ILBC_28        ACCTCG                    CGAGGT
## Even3        Even3  ILBC_29        ACCTGT                    ACAGGT
##         Barcode_full_length SampleType
##                   <factor>   <factor>
## CL3             CTAGCGTGCGT      Soil
## CC1             CATCGACGAGT      Soil
## SV1             GTACGCACAGT      Soil
## M31Fcsw         TCGACATCTCT      Feces
## M11Fcsw         CGACTGCAGCT      Feces
## ...                    ...        ...
## TS28            GCATCGTCTGG      Feces
## TS29            CTAGTCGCTGG      Feces
## Even1           TGACTCTGCGG      Mock
## Even2           TCTGATCGAGG      Mock
## Even3           AGAGAGACAGG      Mock
##                                      Description
##                                         <factor>
## CL3     Calhoun South Carolina Pine soil, pH 4.9
## CC1     Cedar Creek Minnesota, grassland, pH 6.1
## SV1     Sevilleta new Mexico, desert scrub, pH 8.3
## M31Fcsw M3, Day 1, fecal swab, whole body study
## M11Fcsw M1, Day 1, fecal swab, whole body study
## ...                                             ...
## TS28                                       Twin #1
## TS29                                       Twin #2
## Even1                                      Even1
## Even2                                      Even2
```

```
## Even3                                    Even3
```

2.2.3 rowData

`rowData` contains data on the features of the analyzed samples. Of particular interest for the microbiome field this is used to store taxonomic information.

```
rowData(tse)
```

```
## DataFrame with 19216 rows and 7 columns
##            Kingdom        Phylum        Class        Order        Family
##        <character>   <character>  <character>  <character>   <character>
## 549322     Archaea Crenarchaeota Thermoprotei           NA            NA
## 522457     Archaea Crenarchaeota Thermoprotei           NA            NA
## 951        Archaea Crenarchaeota Thermoprotei Sulfolobales Sulfolobaceae
## 244423     Archaea Crenarchaeota        Sd-NA           NA            NA
## 586076     Archaea Crenarchaeota        Sd-NA           NA            NA
## ...            ...           ...          ...          ...           ...
## 278222    Bacteria           SR1           NA           NA            NA
## 463590    Bacteria           SR1           NA           NA            NA
## 535321    Bacteria           SR1           NA           NA            NA
## 200359    Bacteria           SR1           NA           NA            NA
## 271582    Bacteria           SR1           NA           NA            NA
##              Genus               Species
##        <character>           <character>
## 549322          NA                    NA
## 522457          NA                    NA
## 951     Sulfolobus Sulfolobusacidocalda..
## 244423          NA                    NA
## 586076          NA                    NA
## ...            ...                   ...
## 278222          NA                    NA
## 463590          NA                    NA
## 535321          NA                    NA
## 200359          NA                    NA
## 271582          NA                    NA
```

2.2.4 rowTree

Phylogenetic trees also play an important role for the microbiome field. The `TreeSummarizedExperiment` class is able to keep track of feature and node relations via two functions, `rowTree` and `rowLinks`.

A tree can be accessed via `rowTree` as `phylo` object.

```
rowTree(tse)
```

```
##
## Phylogenetic tree with 19216 tips and 19215 internal nodes.
##
## Tip labels:
##   549322, 522457, 951, 244423, 586076, 246140, ...
## Node labels:
##   , 0.858.4, 1.000.154, 0.764.3, 0.995.2, 1.000.2, ...
##
## Rooted; includes branch lengths.
```

The links to the individual features are available through `rowLinks`.

```
rowLinks(tse)
```

```
## LinkDataFrame with 19216 rows and 5 columns
##           nodeLab   nodeNum nodeLab_alias    isLeaf   whichTree
##       <character> <integer>   <character> <logical> <character>
## 1          549322         1       alias_1      TRUE       phylo
## 2          522457         2       alias_2      TRUE       phylo
## 3             951         3       alias_3      TRUE       phylo
## 4          244423         4       alias_4      TRUE       phylo
## 5          586076         5       alias_5      TRUE       phylo
## ...           ...       ...           ...       ...         ...
## 19212      278222     19212   alias_19212      TRUE       phylo
## 19213      463590     19213   alias_19213      TRUE       phylo
## 19214      535321     19214   alias_19214      TRUE       phylo
## 19215      200359     19215   alias_19215      TRUE       phylo
## 19216      271582     19216   alias_19216      TRUE       phylo
```

Please note that there can be a 1:1 relationship between tree nodes and features, but this is not a must have. This means there can be features, which are not linked to nodes, and nodes, which are not linked to features. To change the links in an existing object, the `changeTree` function is available.

2.2.5 Alternative experiments

Alternative experiments differ from transformations as they can contain complementary data, which is no longer tied to the same dimensions as the assay data. However, the number of samples (columns) must be the same.

This can come into play for instance when one has taxonomic abundance profiles quantified with different measurement technologies, such as phylogenetic microarrays, amplicon sequencing, or metagenomic sequencing. Such alternative experiments that concern the same samples can be stored as

1. Separate *assays* assuming that the taxonomic information can be mapped between feature directly 1:1; or
2. data in the *altExp* slot of the `TreeSummarizedExperiment`, if the feature dimensions differ. Each element of the *altExp* slot is a `SummarizedExperiment` or an object from a derived class with independent feature data.

As an example, we show how to store taxonomic abundance tables agglomerated at different taxonomic levels. However, the data could as well originate from entirely different measurement sources as long as the samples are matched.

```
# Agglomerate the data to Phylym level
tse_phylum <- agglomerateByRank(tse, "Phylum")
# both have the same number of columns (samples)
dim(tse)
```

```
## [1] 19216    26
```

```
dim(tse_phylum)
```

```
## [1] 67 26
```

```
# Add the new table as an alternative experiment
altExp(tse, "Phylum") <- tse_phylum
altExpNames(tse)
```

```
## [1] "Phylum"
```

```
# Pick a sample subset: this acts on both altExp and assay data
tse[,1:10]
```

```
## class: TreeSummarizedExperiment
## dim: 19216 10
## metadata(0):
## assays(2): counts relabundance
## rownames(19216): 549322 522457 ... 200359 271582
```

```
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(10): CL3 CC1 ... M31Tong M11Tong
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(1): Phylum
## rowLinks: a LinkDataFrame (19216 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

```
dim(altExp(tse[,1:10],"Phylum"))
```

```
## [1] 67 10
```

For more details of altExp have a look at the Intro vignette of the `SingleCellExperiment` package (Lun and Risso, 2020).

2.2.6 MultiAssayExperiments

Multiple experiments relate to complementary measurement types, such as transcriptomic or metabolomic profiling of the microbiome or the host. Multiple experiments can be represented using the same options as alternative experiments, or by using the `MultiAssayExperiment` class (Ramos et al., 2017). Depending on how the datasets relate to each other the data can be stored as:

1. Separate *altExp* if the samples can be matched directly 1:1; or
2. As `MultiAssayExperiment` objects, in which the connections between samples are defined through a `sampleMap`. Each element on the `experimentsList` of an `MultiAssayExperiment` is `matrix` or `matrix`-like object including `SummarizedExperiment` objects, and the number of samples can differ between the elements.

```
#TODO: Find the right dataset to explain a non 1:1 sample
↪  relationship
```

For information have a look at the intro vignette of the `MultiAssayExperiment` package.

Table 2.1: **Recommended options for storing multiple data tables in microbiome studies** The *assays* are best suited for data transformations (one-to-one match between samples and columns across the assays). The *alternative experiments* are particularly suitable for alternative versions of the data that are of same type but may have a different number of features (e.g. taxonomic groups); this is for instance the case with taxonomic abundance tables agglomerated at different levels (e.g. genus vs. phyla) or alternative profiling technologies (e.g. amplicon sequencing vs. shallow shotgun metagenomics). For alternative experiments one-to-one match between samples (cols) is required but the alternative experiment tables can have different numbers of features (rows). Finally, elements of the *MultiAssayExperiment* provide the most flexible way to incorporate multi-omic data tables with flexible numbers of samples and features. We recommend these conventions as the basis for methods development and application in microbiome studies.

Option	Rows (features)	Cols (samples)	Recommended
assays	match	match	Data transformations
altExp	free	match	Alternative experiments
MultiAssay	free	free (mapping)	Multi-omic experiments

2.3 Loading experimental microbiome data

2.3.1 16S workflow

Result of amplicon sequencing is large number of files that include all the sequences that were read from samples. Those sequences need to be matched with taxa. Additionally, we need to know how many times each taxa were found from each sample.

There are several algorithms to do that, and DADA2 is one of the most common. You can find DADA2 pipeline tutorial for example from here. After DADA2 portion of the tutorial is the data is stored into *phyloseq* object (Bonus: Hand-off to phyloseq). To store the data to *TreeSummarizedExperiment*, follow the example below.

You can find full workflow script without further explanations and comments from here

Load required packages.

```
library(mia)
library(ggplot2)
```

```
if( !require("BiocManager") ){
    install.packages("BiocManager")
    library("BiocManager")
}

if( !require("Biostrings") ){
    BiocManager::install("Biostrings")
    library("Biostrings")
}
library(Biostrings)
```

Create arbitrary example sample metadata like it was done in tutorial. Usually, sample metadata is imported as a file.

```
samples.out <- rownames(seqtab.nochim)
subject <- sapply(strsplit(samples.out, "D"), `[`, 1)
gender <- substr(subject,1,1)
subject <- substr(subject,2,999)
day <- as.integer(sapply(strsplit(samples.out, "D"), `[`, 2))
samdf <- data.frame(Subject=subject, Gender=gender, Day=day)
samdf$When <- "Early"
samdf$When[samdf$Day>100] <- "Late"
rownames(samdf) <- samples.out
```

Convert data into right format and create *TreeSE* object.

```
# Create a list that contains assays
counts <- t(seqtab.nochim)
counts <- as.matrix(counts)
assays <- SimpleList(counts = counts)

# Convert colData and rowData into DataFrame
samdf <- DataFrame(samdf)
taxa <- DataFrame(taxa)

# Create TreeSE
tse <- TreeSummarizedExperiment(assays = assays,
                                colData = samdf,
                                rowData = taxa
                                )

# Remove mock sample like it is also done in DADA2 pipeline
 ↪  tutorial
```

```
tse <- tse[ , colnames(tse) != "mock"]
```

Add sequences into *referenceSeq* slot and convert rownames into simpler format.

```
# Convert sequences into right format
dna <- Biostrings::DNAStringSet( rownames(tse) )
# Add sequences into referenceSeq slot
referenceSeq(tse) <- dna
# Convert rownames into ASV_number format
rownames(tse) <- paste0("ASV", seq( nrow(tse) ))
tse
```

```
## class: TreeSummarizedExperiment
## dim: 232 20
## metadata(0):
## assays(1): counts
## rownames(232): ASV1 ASV2 ... ASV231 ASV232
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(20): F3D0 F3D1 ... F3D9 Mock
## colData names(4): Subject Gender Day When
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
## referenceSeq: a DNAStringSet (232 sequences)
```

2.3.2 Import from external files

Microbiome (taxonomic) profiling data is commonly distributed in various file formats. You can import such external data files as a (Tree)SummarizedExperiment object but the details depend on the file format. Here, we provide examples for common formats.

CSV data tables can be imported with the standard R functions, then converted to the desired format. For detailed examples, you can check the Bioconductor course material by Martin Morgan. The following example reads abundance tables, taxonomic mapping tables, and sample metadata, assuming that the input data files are properly prepared with appropriate row and column names.

```
count_file <- "data/assay_taxa.csv"
tax_file <- "data/rowdata_taxa.csv"
sample_file <- "data/coldata.csv"

# Load files
counts  <- read.csv(count_file)   # Abundance table (e.g. ASV
 ↪  data; to assay data)
tax     <- read.csv(tax_file)     # Taxonomy table (to rowData)
samples <- read.csv(sample_file)  # Sample data (to colData)
```

Always ensure that the tables have rownames! The *TreeSE* constructor compares rownames and makes sure that, for example, right samples are linked with right patient.

```
# Add rownames and remove an additional column
rownames(counts) <- counts$X
counts$X <- NULL

# Add rownames and remove an additional column
rownames(samples) <- samples$X
samples$X <- NULL

# Add rownames and remove an additional column
rownames(tax) <- tax$X
tax$X <- NULL

# As an example:
# If e.g. samples do not match between colData and counts table,
 ↪  you must order
# counts based on colData
if( any( colnames(counts) != rownames(samples) ) ){
    counts <- counts[ , rownames(samples) ]
}

# And same with rowData and counts...
if( any( rownames(counts) != rownames(tax) ) ){
    counts <- counts[ rownames(tax), ]
}
```

The tables must be in correct format:

- counts –> matrix
- rowData –> DataFrame
- colData –> DataFrame

```
# Ensure that the data is in correct format

# counts should be in matrix format
counts <- as.matrix(counts)
# And it should be added to a SimpleList
assays <-  SimpleList(counts = counts)

# colData and rowData should be in DataFrame format
colData <- DataFrame(colData)
rowData <- DataFrame(rowData)

# Create a TreeSE
tse_taxa <- TreeSummarizedExperiment(assays = assays,
                                     colData = samples,
                                     rowData = tax)

tse_taxa
```

```
## class: TreeSummarizedExperiment
## dim: 12706 40
## metadata(0):
## assays(1): counts
## rownames(12706): GAYR01026362.62.2014 CVJT01000011.50.2173 ...
##   JRJTB:03787:02429 JRJTB:03787:02478
## rowData names(7): Phylum Class ... Species OTU
## colnames(40): C1 C2 ... C39 C40
## colData names(6): Sample Rat ... Fat XOS
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

To construct a *MultiAssayExperiment* object, just combine multiple *TreeSE* data containers. Here we import metabolite data from the same study.

```
count_file <- "data/assay_metabolites.csv"
sample_file <- "data/coldata.csv"

# Load files
counts  <- read.csv(count_file)
samples <- read.csv(sample_file)
```

```
# Add rownames and remove an additional column
rownames(counts) <- counts$X
counts$X <- NULL
rownames(samples) <- samples$X
samples$X <- NULL

# Convert into right format
counts <- as.matrix(counts)
assays <-  SimpleList(concs = counts)
colData <- DataFrame(colData)

# Create a TreeSE
tse_metabolite <- TreeSummarizedExperiment(assays = assays,
                                           colData = samples)
tse_metabolite
```

```
## class: TreeSummarizedExperiment
## dim: 38 40
## metadata(0):
## assays(1): concs
## rownames(38): Butyrate Acetate ... Malonate 1,3-dihydroxyacetone
## rowData names(0):
## colnames(40): C1 C2 ... C39 C40
## colData names(6): Sample Rat ... Fat XOS
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

Now we can combine these two experiments into *MAE*.

```
# Create an ExperimentList that includes experiments
experiments <- ExperimentList(microbiome = tse_taxa,
                              metabolite = tse_metabolite)

# Create a MAE
mae <- MultiAssayExperiment(experiments = experiments)

mae
```

```
## A MultiAssayExperiment object of 2 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 2:
##  [1] microbiome: TreeSummarizedExperiment with 12706 rows and 40 columns
##  [2] metabolite: TreeSummarizedExperiment with 38 rows and 40 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

Specific import functions are provided for:

- Biom files (see `help(mia::loadFromBiom)`)
- QIIME2 files (see `help(mia::loadFromQIIME2)`)
- Mothur files (see `help(mia::loadFromMothur)`)

2.3.2.1 Biom example

This example shows how Biom files are imported into a `TreeSummarizedExperiment` object.

The data is from following publication: Tengeler AC *et al.* (2020) **Gut microbiota from persons with attention-deficit/hyperactivity disorder affects the brain in mice**.

The data set consists of 3 files:

- biom file: abundance table and taxonomy information
- csv file: sample metadata
- tree file: phylogenetic tree

Store the data in your desired local directory (for instance, *data/* under the working directory), and define source file paths

```
biom_file_path <- "data/Aggregated_humanization2.biom"
sample_meta_file_path <- "data/Mapping_file_ADHD_aggregated.csv"
tree_file_path <- "data/Data_humanization_phylo_aggregation.tre"
```

Now we can load the biom data into a SummarizedExperiment (SE) object.

```
library(mia)

# Imports the data
se <- loadFromBiom(biom_file_path)

# Check
se
```

```
## class: TreeSummarizedExperiment
## dim: 151 27
## metadata(0):
## assays(1): counts
## rownames(151): 1726470 1726471 ... 17264756 17264757
## rowData names(6): taxonomy1 taxonomy2 ... taxonomy5 taxonomy6
## colnames(27): A110 A111 ... A38 A39
## colData names(0):
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

The `assays` slot includes a list of abundance tables. The imported abundance table is named as "counts". Let us inspect only the first cols and rows.

```
assays(se)$counts[1:3, 1:3]
```

```
##           A110  A111  A12
## 1726470  17722 11630    0
## 1726471  12052     0 2679
## 17264731     0   970    0
```

The `rowdata` includes taxonomic information from the biom file. The `head()` command shows just the beginning of the data table for an overview.

`knitr::kable()` is for printing the information more nicely.

```
head(rowData(se))
```

```
## DataFrame with 6 rows and 6 columns
##             taxonomy1          taxonomy2           taxonomy3
```

```
##           <character>        <character>         <character>
## 1726470  "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 1726471  "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 17264731 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 17264726 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 1726472  "k__Bacteria p__Verrucomicrobia c__Verrucomicrobiae
## 17264724 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
##                      taxonomy4              taxonomy5           taxonomy6
##                    <character>            <character>         <character>
## 1726470       o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 1726471       o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 17264731      o__Bacteroidales  f__Porphyromonadaceae g__Parabacteroides"
## 17264726      o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 1726472  o__Verrucomicrobiales f__Verrucomicrobiaceae     g__Akkermansia"
## 17264724      o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
```

These taxonomic rank names (column names) are not real rank names. Let's replace them with real rank names.

In addition to that, the taxa names include, e.g., '"k___' before the name, so let's make them cleaner by removing them.

```
names(rowData(se)) <- c("Kingdom", "Phylum", "Class", "Order",
                        "Family", "Genus")

# Goes through the whole DataFrame. Removes '.*[kpcofg]__' from
 ↪  strings, where [kpcofg]
# is any character from listed ones, and .* any character.
rowdata_modified <- BiocParallel::bplapply(rowData(se),
                                           FUN =
                                            ↪  stringr::str_remove,
                                            ↪
                                           pattern =
                                            ↪  '.*[kpcofg]__')

# Genus level has additional '\"', so let's delete that also
rowdata_modified <- BiocParallel::bplapply(rowdata_modified,
                                           FUN =
                                            ↪  stringr::str_remove,
                                            ↪
                                           pattern = '\"')

# rowdata_modified is a list, so it is converted back to
 ↪  DataFrame format.
rowdata_modified <- DataFrame(rowdata_modified)
```

```
# And then assigned back to the SE object
rowData(se) <- rowdata_modified

# Now we have a nicer table
head(rowData(se))
```

```
## DataFrame with 6 rows and 6 columns
##              Kingdom          Phylum            Class              Order
##          <character>     <character>      <character>        <character>
## 1726470     Bacteria   Bacteroidetes      Bacteroidia      Bacteroidales
## 1726471     Bacteria   Bacteroidetes      Bacteroidia      Bacteroidales
## 17264731    Bacteria   Bacteroidetes      Bacteroidia      Bacteroidales
## 17264726    Bacteria   Bacteroidetes      Bacteroidia      Bacteroidales
## 1726472     Bacteria Verrucomicrobia Verrucomicrobiae Verrucomicrobiales
## 17264724    Bacteria   Bacteroidetes      Bacteroidia      Bacteroidales
##                        Family           Genus
##                   <character>     <character>
## 1726470       Bacteroidaceae     Bacteroides
## 1726471       Bacteroidaceae     Bacteroides
## 17264731  Porphyromonadaceae Parabacteroides
## 17264726      Bacteroidaceae     Bacteroides
## 1726472  Verrucomicrobiaceae     Akkermansia
## 17264724      Bacteroidaceae     Bacteroides
```

We notice that the imported biom file did not contain the sample meta data yet, so it includes an empty data frame.

```
head(colData(se))
```

```
## DataFrame with 6 rows and 0 columns
```

Let us add a sample metadata file.

```
# We use this to check what type of data it is
# read.table(sample_meta_file_path)

# It seems like a comma separated file and it does not include
 ↪  headers
# Let us read it and then convert from data.frame to DataFrame
# (required for our purposes)
sample_meta <- DataFrame(read.table(sample_meta_file_path, sep =
 ↪  ",", header = FALSE))
```

```
# Add sample names to rownames
rownames(sample_meta) <- sample_meta[,1]

# Delete column that included sample names
sample_meta[,1] <- NULL

# We can add headers
colnames(sample_meta) <- c("patient_status", "cohort",
 ↪  "patient_status_vs_cohort", "sample_name")

# Then it can be added to colData
colData(se) <- sample_meta
```

Now `colData` includes the sample metadata.

```
head(colData(se))
```

```
## DataFrame with 6 rows and 4 columns
##      patient_status      cohort patient_status_vs_cohort sample_name
##         <character> <character>              <character> <character>
## A110           ADHD    Cohort_1            ADHD_Cohort_1        A110
## A12            ADHD    Cohort_1            ADHD_Cohort_1         A12
## A15            ADHD    Cohort_1            ADHD_Cohort_1         A15
## A19            ADHD    Cohort_1            ADHD_Cohort_1         A19
## A21            ADHD    Cohort_2            ADHD_Cohort_2         A21
## A23            ADHD    Cohort_2            ADHD_Cohort_2         A23
```

Now, let's add a phylogenetic tree.

The current data object, se, is a SummarizedExperiment object. This does not include a slot for adding a phylogenetic tree. In order to do this, we can convert the SE object to an extended TreeSummarizedExperiment object which includes also a `rowTree` slot.

TreeSummarizedExperiment contains also other additional slots and features which is why we recommend to use `TreeSE`.

```
tse <- as(se, "TreeSummarizedExperiment")

# tse includes same data as se
tse
```

```
## class: TreeSummarizedExperiment
## dim: 151 27
```

```
## metadata(0):
## assays(1): counts
## rownames(151): 1726470 1726471 ... 17264756 17264757
## rowData names(6): Kingdom Phylum ... Family Genus
## colnames(27): A110 A12 ... A35 A38
## colData names(4): patient_status cohort patient_status_vs_cohort
##   sample_name
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

Next, let us read the tree data file and add it to the R data object (tse).

```
# Reads the tree file
tree <- ape::read.tree(tree_file_path)

# Add tree to rowTree
rowTree(tse) <- tree

# Check
tse
```

```
## class: TreeSummarizedExperiment
## dim: 151 27
## metadata(0):
## assays(1): counts
## rownames(151): 1726470 1726471 ... 17264756 17264757
## rowData names(6): Kingdom Phylum ... Family Genus
## colnames(27): A110 A12 ... A35 A38
## colData names(4): patient_status cohort patient_status_vs_cohort
##   sample_name
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (151 rows)
## rowTree: 1 phylo tree(s) (151 leaves)
## colLinks: NULL
## colTree: NULL
```

Now `rowTree` includes a phylogenetic tree:

```
head(rowTree(tse))
```

2.3.3 Conversions between data formats in R

If the data has already been imported in R in another format, it can be readily converted into `TreeSummarizedExperiment`, as shown in our next example. Note that similar conversion functions to `TreeSummarizedExperiment` are available for multiple data formats via the `mia` package (see makeTreeSummarizedExperimentFrom* for phyloseq, Biom, and DADA2).

```
library(mia)

# phyloseq example data
data(GlobalPatterns, package="phyloseq")
GlobalPatterns_phyloseq <- GlobalPatterns
GlobalPatterns_phyloseq
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

```
# convert phyloseq to TSE
GlobalPatterns_TSE <-
 ↪  makeTreeSummarizedExperimentFromPhyloseq(GlobalPatterns_phyloseq)
 ↪
GlobalPatterns_TSE
```

```
## class: TreeSummarizedExperiment
## dim: 19216 26
## metadata(0):
## assays(1): counts
## rownames(19216): 549322 522457 ... 200359 271582
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (19216 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
```

```
## colLinks: NULL
## colTree: NULL
```

We can also convert `TreeSummarizedExperiment` objects into `phyloseq` with respect to the shared components that are supported by both formats (i.e. taxonomic abundance table, sample metadata, taxonomic table, phylogenetic tree, sequence information). This is useful for instance when additional methods are available for `phyloseq`.

```
# convert TSE to phyloseq
GlobalPatterns_phyloseq2 <-
 ↪  makePhyloseqFromTreeSummarizedExperiment(GlobalPatterns_TSE)
GlobalPatterns_phyloseq2
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

Conversion is possible between other data formats. Interested readers can refer to the following functions: * makeTreeSummarizedExperimentFromDADA2 * makeSummarizedExperimentFromBiom * loadFromMetaphlan * readQZA

2.4 Demonstration data

Open demonstration data for testing and benchmarking purposes is available from multiple locations. This chapter introduces some options. The other chapters of this book provide ample examples about the use of the data.

2.4.1 Package data

The `mia` R package contains example data sets that are direct conversions from the alternative `phyloseq` container to the `TreeSummarizedExperiment` container.

List the available datasets in the `mia` package:

```
library(mia)
data(package="mia")
```

Load the `GlobalPatterns` data from the `mia` package:

```
data("GlobalPatterns", package="mia")
GlobalPatterns
```

```
## class: TreeSummarizedExperiment
## dim: 19216 26
## metadata(0):
## assays(1): counts
## rownames(19216): 549322 522457 ... 200359 271582
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (19216 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

Check the documentation for this data set:

```
## Help on topic 'GlobalPatterns' was found in the following packages:
##
##   Package               Library
##   phyloseq              /__w/_temp/Library
##   mia                   /__w/_temp/Library
##
##
## Using the first match ...
```

2.4.2 ExperimentHub data

ExperimentHub provides a variety of data resources, including the microbiomeDataSets package (Morgan and Shepherd, 2021; Lahti et al., 2021a).

A table of the available data sets is available through the `availableDataSets` function.

```
library(microbiomeDataSets)
availableDataSets()
```

```
##             Dataset
## 1  GrieneisenTSData
```

```
## 2    HintikkaXOData
## 3       LahtiMLData
## 4        LahtiMData
## 5       LahtiWAData
## 6      OKeefeDSData
## 7 SilvermanAGutData
## 8        SongQAData
## 9   SprockettTHData
```

All data are downloaded from ExperimentHub and cached for local re-use. Check the man pages of each function for a detailed documentation of the data contents and references. Let us retrieve a *MultiAssayExperiment* data set:

```
# mae <- HintikkaXOData()
# Since HintikkaXOData is now added to mia, we can load it
 ↪  directly from there
# We suggest to check other datasets from microbiomeDataSets
data(HintikkaXOData)
mae <- HintikkaXOData
```

Data is available in *SummarizedExperiment*, `r Biocpkg("TreeSummarizedExperiment")` and `r Biocpkg("MultiAssayExperiment")` data containers; see the separate page on alternative containers for more details.

2.4.3 Other data sources

The curatedMetagenomicData is an independent source that provides various example data sets as `(Tree)SummarizedExperiment` objects (Pasolli et al., 2017). This resource provides curated human microbiome data including gene families, marker abundance, marker presence, pathway abundance, pathway coverage, and relative abundance for samples from different body sites. See the package homepage for more details on data availability and access.

As one example, let us retrieve the Vatanen (2016) (Vatanen et al., 2016) data set. This is a larger collection with a bit longer download time.

```
library(curatedMetagenomicData)
tse <- curatedMetagenomicData("Vatanen*", dryrun = FALSE, counts
 ↪  = TRUE)
```

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] microbiomeDataSets_1.1.7       phyloseq_1.40.0
 [3] BiocManager_1.30.20            ggplot2_3.4.1
 [5] mia_1.7.11                     MultiAssayExperiment_1.24.0
 [7] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
 [9] XVector_0.38.0                 SingleCellExperiment_1.20.1
[11] SummarizedExperiment_1.28.0    Biobase_2.58.0
[13] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[15] IRanges_2.32.0                 S4Vectors_0.36.2
[17] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[19] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[21] rebook_1.6.0

loaded via a namespace (and not attached):
  [1] AnnotationHub_3.4.0           BiocFileCache_2.4.0
  [3] plyr_1.8.8                    igraph_1.4.1
  [5] lazyeval_0.2.2                splines_4.2.1
  [7] BiocParallel_1.32.6           scater_1.26.1
  [9] digest_0.6.31                 foreach_1.5.2
 [11] yulab.utils_0.0.6             htmltools_0.5.5
 [13] viridis_0.6.2                 fansi_1.0.4
 [15] magrittr_2.0.3                memoise_2.0.1
 [17] ScaledMatrix_1.6.0            cluster_2.1.4
 [19] DECIPHER_2.26.0               colorspace_2.1-0
 [21] blob_1.2.4                    rappdirs_0.3.3
 [23] ggrepel_0.9.3                 xfun_0.38
```

```
 [25] dplyr_1.1.1                   crayon_1.5.2
 [27] RCurl_1.98-1.12               jsonlite_1.8.4
 [29] graph_1.74.0                  survival_3.5-5
 [31] iterators_1.0.14              ape_5.7-1
 [33] glue_1.6.2                    gtable_0.3.3
 [35] zlibbioc_1.44.0               DelayedArray_0.24.0
 [37] BiocSingular_1.14.0           Rhdf5lib_1.18.2
 [39] scales_1.2.1                  DBI_1.1.3
 [41] Rcpp_1.0.10                   xtable_1.8-4
 [43] viridisLite_0.4.1             decontam_1.18.0
 [45] tidytree_0.4.2                bit_4.0.5
 [47] rsvd_1.0.5                    httr_1.4.5
 [49] dir.expiry_1.4.0              ellipsis_0.3.2
 [51] pkgconfig_2.0.3               XML_3.99-0.14
 [53] scuttle_1.8.4                 CodeDepends_0.6.5
 [55] dbplyr_2.3.2                  utf8_1.2.3
 [57] AnnotationDbi_1.58.0          tidyselect_1.2.0
 [59] rlang_1.1.0                   reshape2_1.4.4
 [61] later_1.3.0                   munsell_0.5.0
 [63] BiocVersion_3.15.2            tools_4.2.1
 [65] cachem_1.0.7                  cli_3.6.1
 [67] DirichletMultinomial_1.40.0   generics_0.1.3
 [69] RSQLite_2.3.0                 ExperimentHub_2.4.0
 [71] ade4_1.7-22                   evaluate_0.20
 [73] biomformat_1.24.0             stringr_1.5.0
 [75] fastmap_1.1.1                 yaml_2.3.7
 [77] knitr_1.42                    bit64_4.0.5
 [79] purrr_1.0.1                   KEGGREST_1.36.3
 [81] nlme_3.1-162                  sparseMatrixStats_1.10.0
 [83] mime_0.12                     compiler_4.2.1
 [85] beeswarm_0.4.0                filelock_1.0.2
 [87] curl_5.0.0                    png_0.1-8
 [89] interactiveDisplayBase_1.34.0 treeio_1.22.0
 [91] tibble_3.2.1                  stringi_1.7.12
 [93] lattice_0.20-45               Matrix_1.5-3
 [95] vegan_2.6-4                   permute_0.9-7
 [97] multtest_2.52.0               vctrs_0.6.1
 [99] pillar_1.9.0                  lifecycle_1.0.3
[101] rhdf5filters_1.8.0            BiocNeighbors_1.16.0
[103] data.table_1.14.8             bitops_1.0-7
[105] irlba_2.3.5.1                 httpuv_1.6.9
[107] R6_2.5.1                      promises_1.2.0.1
[109] bookdown_0.33                 gridExtra_2.3
[111] vipor_0.4.5                   codetools_0.2-19
[113] MASS_7.3-58.3                 rhdf5_2.40.0
[115] withr_2.5.0                   GenomeInfoDbData_1.2.9
```

```
[117] mgcv_1.8-42                   parallel_4.2.1
[119] grid_4.2.1                    beachmat_2.14.0
[121] tidyr_1.3.0                   rmarkdown_2.21
[123] DelayedMatrixStats_1.20.0     shiny_1.7.4
[125] ggbeeswarm_0.7.1
```

Chapter 3

Packages

3.1 Package installation

Several R packages provide methods for the analysis and manipulation of `SummarizedExperiment` and related data containers. One of these is `mia`. The installation for this and other packages has the following procedure.

Stable Bioconductor release version can be installed with:

```
BiocManager::install("microbiome/mia")
```

Biocondcuctor development version requires the installation of the latest R beta version, and this is primarily recommended for those who already have solid experience with R/Bioconductor and need access to the latest experimental updates.

```
BiocManager::install("microbiome/mia", version="devel")
```

The bleeding edge (and potentially unstable) development version lives in Github:

```
devtools::install_github("microbiome/mia")
```

3.2 Some available packages

Some of the R packages supporting the *TreeSummarizedExperiment* framework include:

3.2.1 mia family of methods

- mia : generic microbiome analysis tools

- miaViz : microbiome data visualization
- miaSim : microbiome data simulation
- miaTime : microbiome time series analysis

3.2.2 Tree-based methods {sub-tree-methods}

- philr (external, Silverman et al. (2017))
- mia: Microbiome analysis tools (Ernst et al., 2020)
- miaViz: Microbiome analysis specific visualization (Ernst et al., 2022)
- miaSim: Microbiome data simulations (Simsek et al., 2021)
- miaTime: Microbiome time series analysis (Lahti, 2021)

3.2.3 Differential abundance {sub-diff-abund}

- benchdamic for benchmarking differential abundance methods
- ANCOMBC for differential abundance analysis

3.2.4 Manipulation {sub-manipulation}

- MicrobiotaProcess for analyzing microbiome and other ecological data within the tidy framework

3.2.5 Data

- curatedMetagenomicData a large collection of curated human microbiome data sets
- microbiomeDataSets microbiome demo data sets

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS
```

```
Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] BiocStyle_2.24.0 rebook_1.6.0

loaded via a namespace (and not attached):
 [1] bookdown_0.33       dir.expiry_1.4.0    codetools_0.2-19
 [4] XML_3.99-0.14       digest_0.6.31       stats4_4.2.1
 [7] evaluate_0.20       graph_1.74.0        rlang_1.1.0
[10] cli_3.6.1           filelock_1.0.2      rmarkdown_2.21
[13] tools_4.2.1         xfun_0.38           yaml_2.3.7
[16] fastmap_1.1.1       compiler_4.2.1      BiocGenerics_0.44.0
[19] BiocManager_1.30.20 htmltools_0.5.5     CodeDepends_0.6.5
[22] knitr_1.42
```

Part II

Focus Topics

Chapter 4

Data Manipulation

4.1 Tidying and subsetting

4.1.1 Tidy data

For several custom analysis and visualization packages, such as those from `tidyverse`, the `SE` data can be converted to a long data.frame format with `meltAssay`.

```
library(mia)
data(GlobalPatterns, package="mia")
tse <- GlobalPatterns
tse <- transformCounts(tse, MARGIN = "samples",
 ↪  method="relabundance")

molten_tse <- meltAssay(tse,
                        add_row_data = TRUE,
                        add_col_data = TRUE,
                        assay_name = "relabundance")
molten_tse
```

```
## # A tibble: 499,616 x 17
##    FeatureID SampleID relabundance Kingdom Phylum       Class Order Family Genus
##    <fct>     <fct>           <dbl> <chr>   <chr>        <chr> <chr> <chr>  <chr>
##  1 549322    CL3                 0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  2 549322    CC1                 0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  3 549322    SV1                 0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  4 549322    M31Fcsw             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  5 549322    M11Fcsw             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
```

```
##  6 549322    M31Plmr             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  7 549322    M11Plmr             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  8 549322    F21Plmr             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
##  9 549322    M31Tong             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
## 10 549322    M11Tong             0 Archaea Crenarchaeo~ Ther~ <NA>  <NA>   <NA>
## # i 499,606 more rows
## # i 8 more variables: Species <chr>, X.SampleID <fct>, Primer <fct>,
## #   Final_Barcode <fct>, Barcode_truncated_plus_T <fct>,
## #   Barcode_full_length <fct>, SampleType <fct>, Description <fct>
```

4.1.2 Subsetting

Subsetting data helps to draw the focus of analysis on particular sets of samples and / or features. When dealing with large data sets, the subset of interest can be extracted and investigated separately. This might improve performance and reduce the computational load.

Load:

- mia
- dplyr
- knitr
- data `GlobalPatterns`

Let us store `GlobalPatterns` into `tse` and check its original number of features (rows) and samples (columns). **Note**: when subsetting by sample, expect the number of columns to decrease; when subsetting by feature, expect the number of rows to decrease.

```
# Store data into se and check dimensions
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
# Show dimensions (features x samples)
dim(tse)
```

```
## [1] 19216    26
```

4.1.2.1 Subset by sample (column-wise)

For the sake of demonstration, here we will extract a subset containing only the samples of human origin (feces, skin or tongue), stored as `SampleType` within `colData(tse)` and also in `tse`.

First, we would like to see all the possible values that `SampleType` can take on and how frequent those are:

.	Freq
Feces	4
Freshwater	2
Freshwater (creek)	3
Mock	3
Ocean	3
Sediment (estuary)	3
Skin	3
Soil	3
Tongue	2

```
# Inspect possible values for SampleType
unique(tse$SampleType)
```

```
## [1] Soil              Feces             Skin              Tongue
## [5] Freshwater        Freshwater (creek) Ocean            Sediment (estuary)
## [9] Mock
## 9 Levels: Feces Freshwater Freshwater (creek) Mock ... Tongue
```

```
# Show the frequency of each value
tse$SampleType %>% table()
```

Note: after subsetting, expect the number of columns to equal the sum of the frequencies of the samples that you are interested in. For instance, `ncols = Feces + Skin + Tongue = 4 + 3 + 2 = 9`.

Next, we *logical index* across the columns of `tse` (make sure to leave the first index empty to select all rows) and filter for the samples of human origin. For this, we use the information on the samples from the meta data `colData(tse)`.

```
# Subset by sample
tse_subset_by_sample <- tse[ , tse$SampleType %in% c("Feces",
 ↪  "Skin", "Tongue")]

# Show dimensions
dim(tse_subset_by_sample)
```

```
## [1] 19216     9
```

As a sanity check, the new object `tse_subset_by_sample` should have the original number of features (rows) and a number of samples (columns) equal to the sum of the samples of interest (in this case 9).

Several characteristics can be used to subset by sample:

- origin
- sampling time
- sequencing method
- DNA / RNA barcode
- cohort

4.1.2.2 Subset by feature (row-wise)

Similarly, here we will extract a subset containing only the features that belong to the phyla Actinobacteria and Chlamydiae, stored as `Phylum` within `rowData(tse)`. However, subsetting by feature implies a few more obstacles, such as the presence of `NA` elements and the possible need for agglomeration.

As previously, we would first like to see all the possible values that `Phylum` can take on and how frequent those are:

```
# Inspect possible values for phylum
unique(rowData(tse)$Phylum)
```

```
##  [1] "Crenarchaeota"    "Euryarchaeota"    "Actinobacteria"   "Spirochaetes"
##  [5] "MVP-15"           "Proteobacteria"   "SBR1093"          "Fusobacteria"
##  [9] "Tenericutes"      "ZB3"              "Cyanobacteria"    "GOUTA4"
## [13] "TG3"              "Chlorobi"         "Bacteroidetes"    "Caldithrix"
## [17] "KSB1"             "SAR406"           "LCP-89"           "Thermi"
## [21] "Gemmatimonadetes" "Fibrobacteres"    "GN06"             "AC1"
```

```
## [25] "TM6"              "OP8"              "Elusimicrobia"    "NC10"
## [29] "SPAM"             NA                 "Acidobacteria"    "CCM11b"
## [33] "Nitrospirae"      "NKB19"            "BRC1"             "Hyd24-12"
## [37] "WS3"              "PAUC34f"          "GN04"             "GN12"
## [41] "Verrucomicrobia" "Lentisphaerae"    "LD1"              "Chlamydiae"
## [45] "OP3"              "Planctomycetes"   "Firmicutes"       "OP9"
## [49] "WPS-2"            "Armatimonadetes"  "SC3"              "TM7"
## [53] "GN02"             "SM2F11"           "ABY1_OD1"         "ZB2"
## [57] "OP11"             "Chloroflexi"      "SC4"              "WS1"
## [61] "GAL15"            "AD3"              "WS2"              "Caldiserica"
## [65] "Thermotogae"      "Synergistetes"    "SR1"
```

```
# Show the frequency of each value
rowData(tse)$Phylum %>% table()
```

Note: after subsetting, expect the number of columns to equal the sum of the frequencies of the feature(s) that you are interested in. For instance, `nrows = Actinobacteria + Chlamydiae = 1631 + 21 = 1652`.

Depending on your research question, you might or might not need to agglomerate the data in the first place: if you want to find the abundance of each and every feature that belongs to Actinobacteria and Chlamydiae, agglomeration is not needed; if you want to find the total abundance of all features that belong to Actinobacteria or Chlamydiae, agglomeration is recommended.

4.1.2.2.1 Non-agglomerated data

Next, we *logical index* across the rows of `tse` (make sure to leave the second index empty to select all columns) and filter for the features that fall in either Actinobacteria or Chlamydiae group. For this, we use the information on the samples from the metadata `rowData(tse)`.

The first term with the `%in%` operator includes all the features of interest, whereas the second term after the AND operator `&` filters out all features that have an `NA` in place of the phylum variable.

```
# Subset by feature
tse_subset_by_feature <- tse[rowData(tse)$Phylum %in%
 ↪  c("Actinobacteria", "Chlamydiae") &
 ↪  !is.na(rowData(tse)$Phylum), ]

# Show dimensions
dim(tse_subset_by_feature)
```

```
## [1] 1652   26
```

.	Freq
ABY1_OD1	7
AC1	1
Acidobacteria	1021
Actinobacteria	1631
AD3	9
Armatimonadetes	61
Bacteroidetes	2382
BRC1	13
Caldiserica	3
Caldithrix	10
CCM11b	2
Chlamydiae	21
Chlorobi	64
Chloroflexi	437
Crenarchaeota	106
Cyanobacteria	393
Elusimicrobia	31
Euryarchaeota	102
Fibrobacteres	7
Firmicutes	4356

As a sanity check, the new object, `tse_subset_by_feature`, should have the original number of samples (columns) and a number of features (rows) equal to the sum of the features of interest (in this case, 1652).

4.1.2.2.2 Agglomerated data When total abundances of certain phyla are of relevance, the data is initially agglomerated by Phylum. Then, similar steps as in the case of non-agglomerated data are followed.

```
# Agglomerate by phylum
tse_phylum <- tse %>% agglomerateByRank(rank = "Phylum")

# Subset by feature and remove NAs
tse_phylum_subset_by_feature <-
 ↪  tse_phylum[rowData(tse_phylum)$Phylum %in%
 ↪  c("Actinobacteria", "Chlamydiae") &
 ↪  !is.na(rowData(tse_phylum)$Phylum), ]

# Show dimensions
dim(tse_phylum_subset_by_feature)
```

```
## [1]  2 26
```

Note: as data was agglomerated, the number of rows should equal the number of phyla used to index (in this case, just 2).

Alternatively:

```
# Store features of interest into phyla
phyla <- c("Phylum:Actinobacteria", "Phylum:Chlamydiae")
# subset by feature
tse_phylum_subset_by_feature <- tse_phylum[phyla, ]
# Show dimensions
dim(tse_subset_by_feature)
```

```
## [1] 1652   26
```

The code above returns the non-agglomerated version of the data.

Fewer characteristics can be used to subset by feature:

- Taxonomic rank
- Meta-taxonomic group

For subsetting by kingdom, agglomeration does not apply, whereas for the other ranks it can be applied if necessary.

4.1.2.3 Subset by sample and feature

Finally, we can subset data by sample and feature at once. The resulting subset contains all the samples of human origin and all the features of phyla Actinobacteria or Chlamydiae.

```
# Subset by sample and feature and remove NAs
tse_subset_by_sample_feature <- tse[rowData(tse)$Phylum %in%
 ↪  c("Actinobacteria", "Chlamydiae") &
 ↪  !is.na(rowData(tse)$Phylum), tse$SampleType %in% c("Feces",
 ↪  "Skin", "Tongue")]

# Show dimensions
dim(tse_subset_by_sample_feature)
```

```
## [1] 1652    9
```

Note: the dimensions of `tse_subset_by_sample_feature` agree with those of the previous subsets (9 columns filtered by sample and 1652 rows filtered by feature).

If a study was to consider and quantify the presence of Actinobacteria as well as Chlamydiae in different sites of the human body, `tse_subset_by_sample_feature` might be a suitable subset to start with.

4.1.2.4 Remove empty columns and rows

Sometimes data might contain, e.g., features that are not present in any of the samples. This can occur, for example, after the data subsetting. In certain analyses, we might want to remove those instances.

```
# Agglomerate data at Genus level
tse_genus <- agglomerateByRank(tse, rank = "Genus")
# List bacteria that we want to include
genera <- c("Class:Thermoprotei", "Genus:Sulfolobus",
 ↪  "Genus:Sediminicola")
# Subset data
tse_genus_sub <- tse_genus[genera, ]

tse_genus_sub
```

```
## class: TreeSummarizedExperiment
## dim: 3 26
```

```
## metadata(1): agglomerated_by_rank
## assays(1): counts
## rownames(3): Class:Thermoprotei Genus:Sulfolobus Genus:Sediminicola
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (3 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

```
# List total counts of each sample
colSums(assay(tse_genus_sub, "counts"))
```

```
##      CL3      CC1      SV1  M31Fcsw  M11Fcsw  M31Plmr  M11Plmr  F21Plmr
##        1        0        0        1        1        0        4        1
##  M31Tong  M11Tong LMEpi24M SLEpi20M   AQC1cm   AQC4cm   AQC7cm      NP2
##        7        3        0        2       64      105      136      222
##      NP3      NP5  TRRsed1  TRRsed2  TRRsed3     TS28     TS29    Even1
##     6433     1154        2        2        2        0        0        0
##    Even2    Even3
##        2        0
```

Now we can see that certain samples do not include any bacteria. We can remove those.

```
# Remove samples that do not contain any bacteria
tse_genus_sub <- tse_genus_sub[ , colSums(assay(tse_genus_sub,
 ↪  "counts")) != 0 ]
tse_genus_sub
```

```
## class: TreeSummarizedExperiment
## dim: 3 18
## metadata(1): agglomerated_by_rank
## assays(1): counts
## rownames(3): Class:Thermoprotei Genus:Sulfolobus Genus:Sediminicola
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(18): CL3 M31Fcsw ... TRRsed3 Even2
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
```

```
## altExpNames(0):
## rowLinks: a LinkDataFrame (3 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

The same action can also be applied to the features.

```
# Take only those samples that are collected from feces, skin, or
 ↪  tongue
tse_genus_sub <- tse_genus[ , colData(tse_genus)$SampleType %in%
 ↪  c("Feces", "Skin", "Tongue")]

tse_genus_sub
```

```
## class: TreeSummarizedExperiment
## dim: 1516 9
## metadata(1): agglomerated_by_rank
## assays(1): counts
## rownames(1516): Class:Thermoprotei Genus:Sulfolobus ...
##   Genus:Coprothermobacter Phylum:SR1
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(9): M31Fcsw M11Fcsw ... TS28 TS29
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (1516 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

```
# What is the number of bacteria that are not present?
sum(rowSums(assay(tse_genus_sub, "counts")) == 0)
```

```
## [1] 435
```

We can see that there are bacteria that are not present in these samples we chose. We can remove those bacteria from the data.

```
# Take only those bacteria that are present
tse_genus_sub <- tse_genus_sub[rowSums(assay(tse_genus_sub,
 ↪  "counts")) > 0, ]
```

```
tse_genus_sub
```

```
## class: TreeSummarizedExperiment
## dim: 1081 9
## metadata(1): agglomerated_by_rank
## assays(1): counts
## rownames(1081): Genus:Sulfolobus Order:NRP-J ...
##   Genus:Coprothermobacter Phylum:SR1
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(9): M31Fcsw M11Fcsw ... TS28 TS29
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (1081 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

4.1.3 Splitting

You can split the data based on variables by using the functions `splitByRanks` and `splitOn`.

`splitByRanks` splits the data based on taxonomic ranks. Since the elements of the output list share columns, they can be stored into `altExp`.

```
altExps(tse) <- splitByRanks(tse)
altExps(tse)
```

```
## List of length 7
## names(7): Kingdom Phylum Class Order Family Genus Species
```

If you want to split the data based on another variable than taxonomic rank, use `splitOn`. It works for row-wise and column-wise splitting.

```
splitOn(tse, "SampleType")
```

```
## List of length 9
## names(9): Soil Feces Skin Tongue ... Ocean Sediment (estuary) Mock
```

4.2 Merge data

`mia` package has `mergeSEs` function that merges multiple `SummarizedExperiment` objects. For example, it is possible to combine multiple `TreeSE` objects which each includes one sample.

`mergeSEs` works like `dplyr` joining functions. In fact, there are available `dplyr-like` aliases of `mergeSEs`, such as `full_join`.

```
# Take subsets for demonstration purposes
tse1 <- tse[, 1]
tse2 <- tse[, 2]
tse3 <- tse[, 3]
tse4 <- tse[1:100, 4]
```

```
# With inner join, we want to include all shared rows. When using
 ↪  mergeSEs function
# all samples are always preserved.
tse <- mergeSEs(list(tse1, tse2, tse3, tse4), join = "inner")
tse
```

```
## class: TreeSummarizedExperiment
## dim: 100 4
## metadata(0):
## assays(1): counts
## rownames(100): 239672 243675 ... 549322 951
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(4): CC1 CL3 M31Fcsw SV1
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (100 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

```
# Left join preserves all rows of the 1st object
tse <- mia::left_join(tse1, tse4, missing_values = 0)
tse
```

```
## class: TreeSummarizedExperiment
## dim: 19216 2
```

```
## metadata(0):
## assays(1): counts
## rownames(19216): 239672 243675 ... 146168 594324
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(2): CL3 M31Fcsw
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (19216 rows)
## rowTree: 1 phylo tree(s) (19216 leaves)
## colLinks: NULL
## colTree: NULL
```

4.2.1 Additional functions

- mapTaxonomy
- mergeRows/mergeCols

Chapter 5

Exploration and quality Control

This chapter focuses on the quality control and exploration of microbiome data and establishes commonly used descriptive summaries. Familiarizing with the peculiarities of a given data set is the essential basis for any data analysis and model building.

The dataset should not suffer from severe technical biases, and you should at least be aware of potential challenges, such as outliers, biases, unexpected patterns and so forth. Standard summaries and visualizations can help, and the rest comes with experience. The exploration and quality control can be iterative processes.

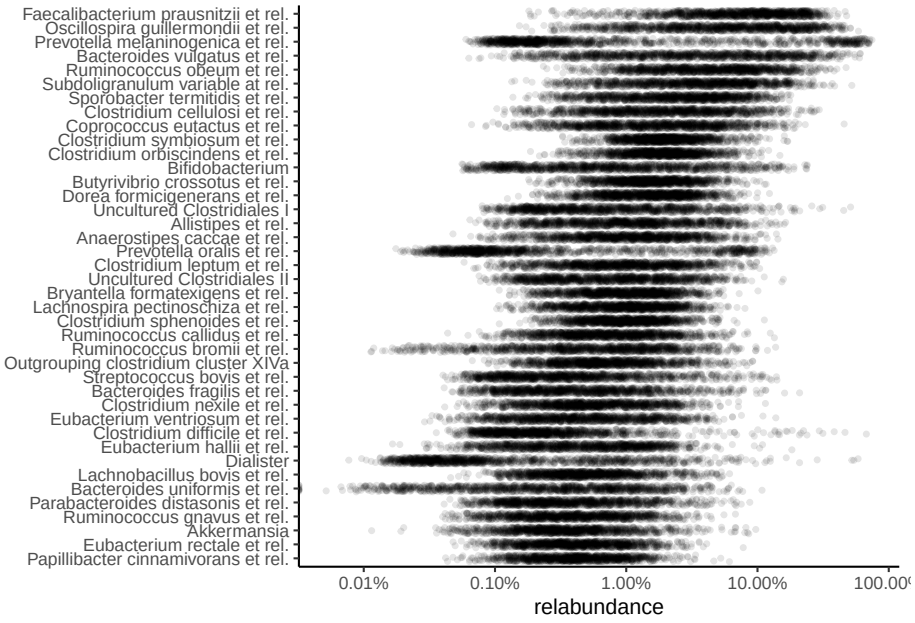
```
library(mia)
```

5.1 Abundance

Abundance visualization is an important data exploration approach. `miaViz` offers the function `plotAbundanceDensity` to plot the most abundant taxa with several options.

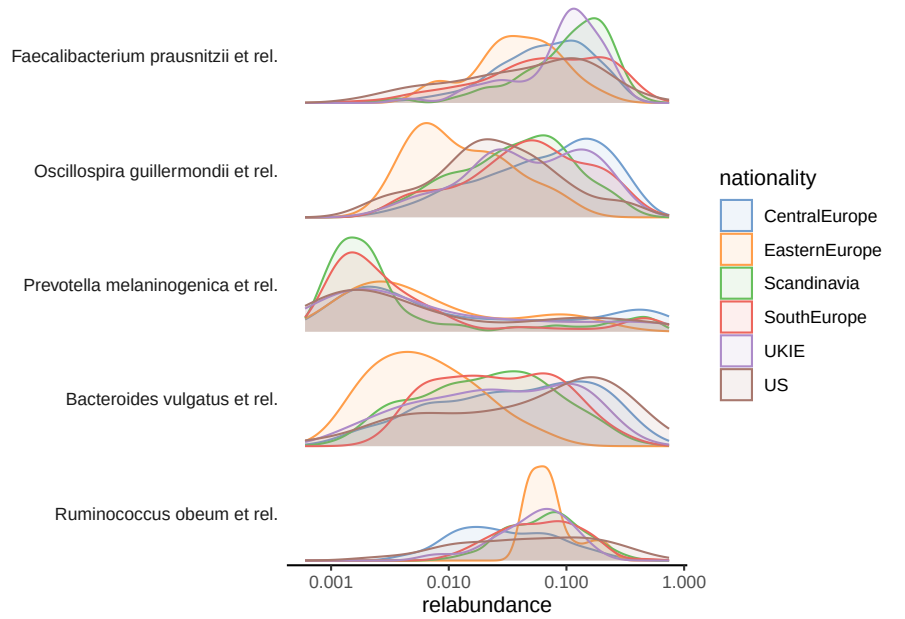
In the following few demonstrations are shown, using the (Lahti et al., 2014) dataset. A Jitter plot based on relative abundance data, similar to the one presented at (Salosensaari et al., 2021) supplementary figure 1, could be visualized as follows:

```
# Loading example data
library(miaTime)
library(miaViz)
data(hitchip1006)
tse <- hitchip1006

# Add relative abundances
tse <- transformCounts(tse, MARGIN = "samples", method =
 ↪  "relabundance")

library(miaViz)
# Use argument names
# assay_name / assay_name / assay_name
# depending on the mia package version
plotAbundanceDensity(tse, layout = "jitter", assay_name =
 ↪  "relabundance",
                     n = 40, point_size=1, point_shape=19,
                      ↪  point_alpha=0.1) +
                     scale_x_log10(label=scales::percent)
```



The relative abundance values for the top-5 taxonomic features can be visualized as a density plot over a log scaled axis, with “nationality” indicated by colors:

```
plotAbundanceDensity(tse, layout = "density", assay_name =
 ↪  "relabundance",
                      n = 5, colour_by="nationality",
                       ↪  point_alpha=1/10) +
    scale_x_log10()
```



5.2 Prevalence

Prevalence quantifies the frequency of samples where certain microbes were detected (above a given detection threshold). The prevalence can be given as sample size (N) or percentage (unit interval).

Investigating prevalence allows you either to focus on changes which pertain to the majority of the samples, or identify rare microbes, which may be *conditionally abundant* in a small number of samples, however.

The population prevalence (frequency) at a 1% relative abundance threshold (`detection = 1/100` and `as_relative = TRUE`), can look like this.

```
head(getPrevalence(tse, detection = 1/100, sort = TRUE,
 ↪  as_relative = TRUE))
```

```
## Faecalibacterium prausnitzii et rel.          Ruminococcus obeum et rel.
```

```
##                                0.9522                                0.9140
##   Oscillospira guillermondii et rel.       Clostridium symbiosum et rel.
##                                0.8801                                0.8714
##     Subdoligranulum variable at rel.    Clostridium orbiscindens et rel.
##                                0.8358                                0.8315
```

The function arguments `detection` and `as_relative` can also be used to access, how many samples do pass a threshold for raw counts. Here the population prevalence (frequency) at the absolute abundance threshold (`as_relative = FALSE`) at read count 1 (`detection = 1`) is accessed.

```
head(getPrevalence(tse, detection = 1, sort = TRUE, assay_name =
 ↪  "counts",
                   as_relative = FALSE))
```

```
##            Uncultured Mollicutes      Uncultured Clostridiales II
##                                1                                1
##       Uncultured Clostridiales I               Tannerella et rel.
##                                1                                1
##   Sutterella wadsworthia et rel. Subdoligranulum variable at rel.
##                                1                                1
```

If the output should be used for subsetting or storing the data in the `rowData`, set `sort = FALSE`.

5.2.1 Prevalence analysis

To investigate microbiome prevalence at a selected taxonomic level, two approaches are available.

First the data can be agglomerated to the taxonomic level and `getPrevalence` applied on the resulting object.

```
# Agglomerate taxa abundances to Phylum level, and add the new
 ↪  table
# to the altExp slot
altExp(tse,"Phylum") <- agglomerateByRank(tse, "Phylum")
# Check prevalence for the Phylum abundance table from the altExp
 ↪  slot
head(getPrevalence(altExp(tse,"Phylum"), detection = 1/100, sort
 ↪  = TRUE,
                   assay_name = "counts", as_relative = TRUE))
```

```
##      Firmicutes   Bacteroidetes  Actinobacteria  Proteobacteria Verrucomicrobia
##       1.0000000       0.9852302       0.4821894       0.2988705       0.1277150
##   Cyanobacteria
##       0.0008688
```

Alternatively, the `rank` argument could be set to perform the agglomeration on the fly.

```
head(getPrevalence(tse, rank = "Phylum", detection = 1/100, sort
 ↪  = TRUE,
                   assay_name = "counts", as_relative = TRUE))
```

```
##      Firmicutes   Bacteroidetes  Actinobacteria  Proteobacteria Verrucomicrobia
##       1.0000000       0.9852302       0.4821894       0.2988705       0.1277150
##   Cyanobacteria
##       0.0008688
```

Note that, by default, `na.rm = TRUE` is used for agglomeration in `getPrevalence`, whereas the default for `agglomerateByRank` is `FALSE` to prevent accidental data loss.

If you only need the names of the prevalent taxa, `getPrevalentTaxa` is available. This returns the taxa that exceed the given prevalence and detection thresholds.

```
getPrevalentTaxa(tse, detection = 0, prevalence = 50/100)
prev <- getPrevalentTaxa(tse, detection = 0, prevalence = 50/100,
                         rank = "Phylum", sort = TRUE)
prev
```

Note that the `detection` and `prevalence` thresholds are not the same, since `detection` can be applied to relative counts or absolute counts depending on whether `as_relative` is set `TRUE` or `FALSE`

The function 'getPrevalentAbundance' can be used to check the total relative abundance of the prevalent taxa (between 0 and 1).

5.2.2 Rare taxa

Related functions are available for the analysis of rare taxa (`rareMembers`; `rareAbundance`; `lowAbundance`, `getRareTaxa`, `subsetByRareTaxa`).

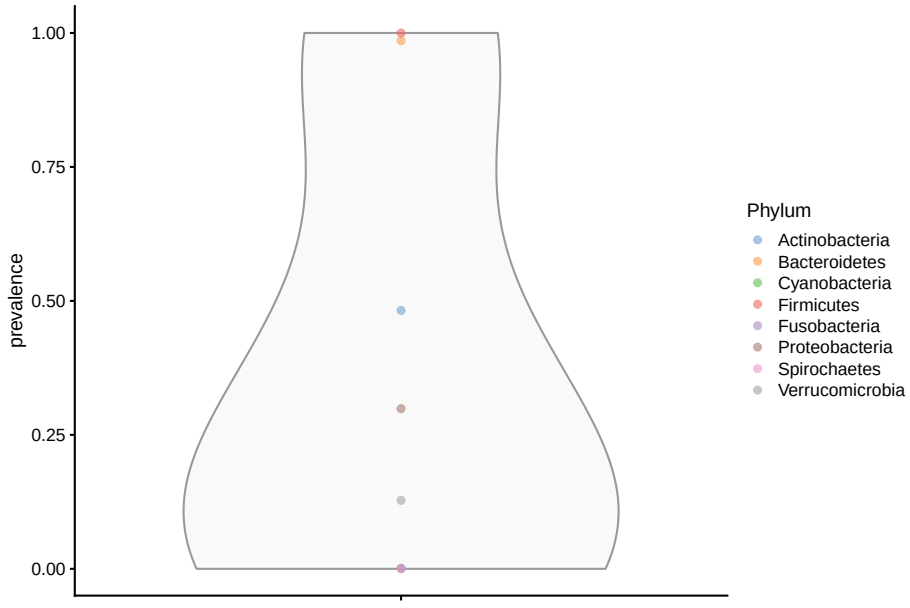
5.2.3 Plotting prevalence

To plot the prevalence, add the prevalence of each taxa to the `rowData`. Here, we are analysing the Phylum level abundances, which are stored in the altExp slot.

```
rowData(altExp(tse,"Phylum"))$prevalence <-
    getPrevalence(altExp(tse,"Phylum"), detection = 1/100, sort =
     ↪  FALSE,
                  assay_name = "counts", as_relative = TRUE)
```

The prevalences can be then plotted via the plotting functions from the `scater` package.

```
library(scater)
plotRowData(altExp(tse,"Phylum"), "prevalence", colour_by =
 ↪  "Phylum")
```



The prevalence can be also visualized on the taxonomic tree with the `miaViz` package.

```
altExps(tse) <- splitByRanks(tse)
altExps(tse) <-
```

```
   lapply(altExps(tse),
          function(y){
              rowData(y)$prevalence <-
                  getPrevalence(y, detection = 1/100, sort =
                   ↪ FALSE,
                                assay_name = "counts",
                                 ↪ as_relative = TRUE)
              y
          })
top_phyla <- getTopTaxa(altExp(tse,"Phylum"),
                        method="prevalence",
                        top=5L,
                        assay_name="counts")
top_phyla_mean <- getTopTaxa(altExp(tse,"Phylum"),
                             method="mean",
                             top=5L,
                             assay_name="counts")
x <- unsplitByRanks(tse, ranks = taxonomyRanks(tse)[1:6])
x <- addTaxonomyTree(x)
```

After some preparation the data is assembled and can be plotted via `plotRowTree`.

```
library(miaViz)
plotRowTree(x[rowData(x)$Phylum %in% top_phyla,],
            edge_colour_by = "Phylum",
            tip_colour_by = "prevalence",
            node_colour_by = "prevalence")
```

```
plotRowTree(x[rowData(x)$Phylum %in% top_phyla_mean,],
            edge_colour_by = "Phylum",
            tip_colour_by = "prevalence",
            node_colour_by = "prevalence")
```

5.3 Quality control

Next, let us load the `GlobalPatterns` data set to illustrate standard microbiome data summaries.

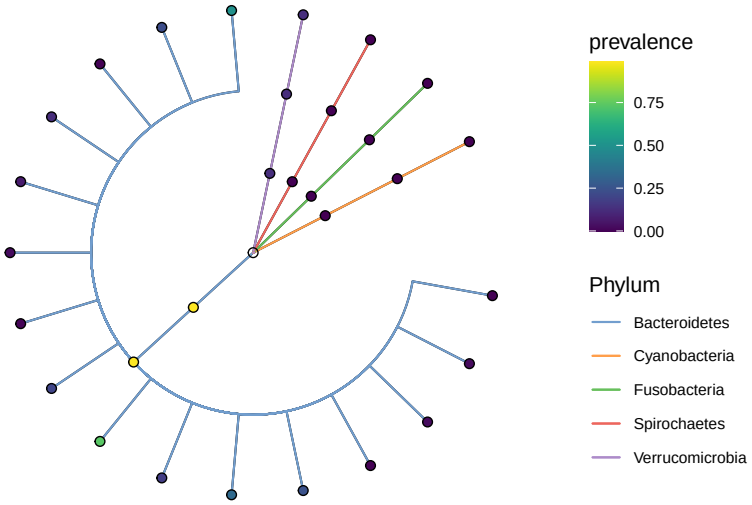


Figure 5.1: Prevalence of top phyla as judged by prevalence

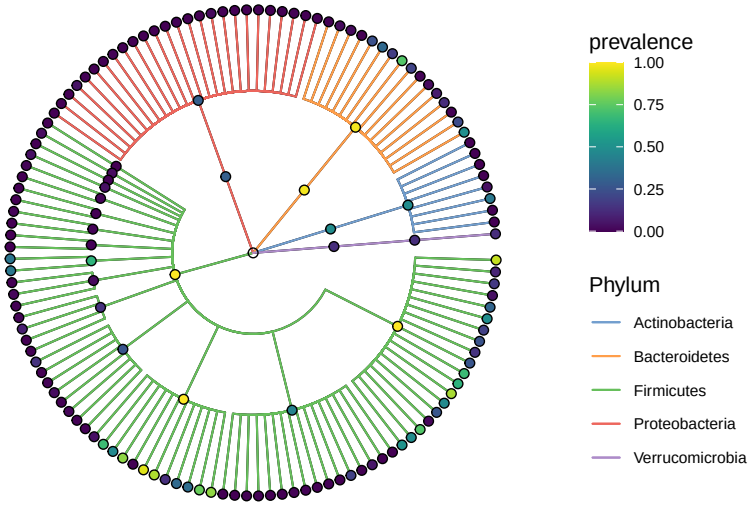


Figure 5.2: Prevalence of top phyla as judged by mean abundance

```
library(mia)
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
```

5.3.1 Top taxa

The `getTopTaxa` identifies top taxa in the data.

```
# Pick the top taxa
top_features <- getTopTaxa(tse, method="median", top=10)

# Check the information for these
rowData(tse)[top_features, taxonomyRanks(tse)]
```

```
## DataFrame with 10 rows and 7 columns
##            Kingdom         Phylum               Class             Order
##        <character>    <character>         <character>       <character>
## 549656    Bacteria  Cyanobacteria         Chloroplast     Stramenopiles
## 331820    Bacteria  Bacteroidetes         Bacteroidia     Bacteroidales
## 317182    Bacteria  Cyanobacteria         Chloroplast     Stramenopiles
## 94166     Bacteria Proteobacteria Gammaproteobacteria    Pasteurellales
## 279599    Bacteria  Cyanobacteria    Nostocophycideae        Nostocales
## 158660    Bacteria  Bacteroidetes         Bacteroidia     Bacteroidales
## 329744    Bacteria Actinobacteria      Actinobacteria   Actinomycetales
## 326977    Bacteria Actinobacteria      Actinobacteria Bifidobacteriales
## 248140    Bacteria  Bacteroidetes         Bacteroidia     Bacteroidales
## 550960    Bacteria Proteobacteria Gammaproteobacteria Enterobacteriales
##                  Family           Genus                Species
##              <character>     <character>            <character>
## 549656                NA              NA                     NA
## 331820     Bacteroidaceae     Bacteroides                     NA
## 317182                NA              NA                     NA
## 94166     Pasteurellaceae     Haemophilus Haemophilusparainflu..
## 279599        Nostocaceae  Dolichospermum                     NA
## 158660     Bacteroidaceae     Bacteroides                     NA
## 329744             ACK-M1              NA                     NA
## 326977 Bifidobacteriaceae Bifidobacterium Bifidobacteriumadole..
## 248140     Bacteroidaceae     Bacteroides      Bacteroidescaccae
## 550960 Enterobacteriaceae     Providencia                     NA
```

5.3.2 Library size / read count

The total counts/sample can be calculated using the `perCellQCMetrics`/`addPerCellQC` from the `scater` package. The former one just calculates the values, whereas the latter one directly adds them to the `colData`.

```
library(scater)
perCellQCMetrics(tse)
```

```
## DataFrame with 26 rows and 3 columns
##               sum  detected     total
##         <numeric> <numeric> <numeric>
## CL3        864077      6964    864077
## CC1       1135457      7679   1135457
## SV1        697509      5729    697509
## M31Fcsw   1543451      2667   1543451
## M11Fcsw   2076476      2574   2076476
## ...           ...       ...       ...
## TS28       937466      2679    937466
## TS29      1211071      2629   1211071
## Even1     1216137      4213   1216137
## Even2      971073      3130    971073
## Even3     1078241      2776   1078241
```

```
tse <- addPerCellQC(tse)
colData(tse)
```

```
## DataFrame with 26 rows and 10 columns
##         X.SampleID   Primer Final_Barcode Barcode_truncated_plus_T
##           <factor> <factor>      <factor>                  <factor>
## CL3          CL3      ILBC_01        AACGCA                    TGCGTT
## CC1          CC1      ILBC_02        AACTCG                    CGAGTT
## SV1          SV1      ILBC_03        AACTGT                    ACAGTT
## M31Fcsw    M31Fcsw  ILBC_04        AAGAGA                    TCTCTT
## M11Fcsw    M11Fcsw  ILBC_05        AAGCTG                    CAGCTT
## ...            ...      ...           ...                       ...
## TS28         TS28   ILBC_25        ACCAGA                    TCTGGT
## TS29         TS29   ILBC_26        ACCAGC                    GCTGGT
## Even1        Even1  ILBC_27        ACCGCA                    TGCGGT
## Even2        Even2  ILBC_28        ACCTCG                    CGAGGT
## Even3        Even3  ILBC_29        ACCTGT                    ACAGGT
##         Barcode_full_length SampleType
##                    <factor>   <factor>
## CL3             CTAGCGTGCGT      Soil
```

```
## CC1             CATCGACGAGT      Soil
## SV1             GTACGCACAGT      Soil
## M31Fcsw         TCGACATCTCT      Feces
## M11Fcsw         CGACTGCAGCT      Feces
## ...                    ...        ...
## TS28            GCATCGTCTGG      Feces
## TS29            CTAGTCGCTGG      Feces
## Even1           TGACTCTGCGG      Mock
## Even2           TCTGATCGAGG      Mock
## Even3           AGAGAGACAGG      Mock
##                                      Description       sum  detected
##                                         <factor> <numeric> <numeric>
## CL3     Calhoun South Carolina Pine soil, pH 4.9      864077      6964
## CC1     Cedar Creek Minnesota, grassland, pH 6.1     1135457      7679
## SV1     Sevilleta new Mexico, desert scrub, pH 8.3    697509      5729
## M31Fcsw M3, Day 1, fecal swab, whole body study      1543451      2667
## M11Fcsw M1, Day 1, fecal swab, whole body study      2076476      2574
## ...                                                      ...       ...       ...
## TS28                                              Twin #1    937466      2679
## TS29                                              Twin #2   1211071      2629
## Even1                                             Even1     1216137      4213
## Even2                                             Even2      971073      3130
## Even3                                             Even3     1078241      2776
##             total
##         <numeric>
## CL3        864077
## CC1       1135457
## SV1        697509
## M31Fcsw   1543451
## M11Fcsw   2076476
## ...           ...
## TS28       937466
## TS29      1211071
## Even1     1216137
## Even2      971073
## Even3     1078241
```

The distribution of calculated library sizes can be visualized as a histogram (left), or by sorting the samples by library size (right).

```
library(ggplot2)

p1 <- ggplot(as.data.frame(colData(tse))) +
        geom_histogram(aes(x = sum), color = "black", fill =
         ↪  "gray", bins = 30) +
```

```
        labs(x = "Library size", y = "Frequency (n)") +
        # scale_x_log10(breaks = scales::trans_breaks("log10",
         ↪  function(x) 10^x),
        # labels = scales::trans_format("log10",
         ↪  scales::math_format(10^.x))) +
        theme_bw() +
        theme(panel.grid.major = element_blank(), # Removes the
         ↪  grid
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black")) # Adds
           ↪  y-axis

library(dplyr)
df <- as.data.frame(colData(tse)) %>%
        arrange(sum) %>%
        mutate(index = 1:n())
p2 <- ggplot(df, aes(y = index, x = sum/1e6)) +
        geom_point() +
        labs(x = "Library size (million reads)", y = "Sample
         ↪  index") +
        theme_bw() +
        theme(panel.grid.major = element_blank(), # Removes the
         ↪  grid
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black")) # Adds
           ↪  y-axis

library(patchwork)
p1 + p2
```

Library sizes - and other variables from `colData` - can be also visualized by using specified function called `plotColData`.

```
library(ggplot2)
# Sort samples by read count, order the factor levels, and store
 ↪  back to tse as DataFrame
# TODO: plotColData could include an option for sorting samples
 ↪  based on colData variables
colData(tse) <- as.data.frame(colData(tse)) %>%
                arrange(X.SampleID) %>%
            mutate(X.SampleID = factor(X.SampleID,
             ↪  levels=X.SampleID)) %>%
```

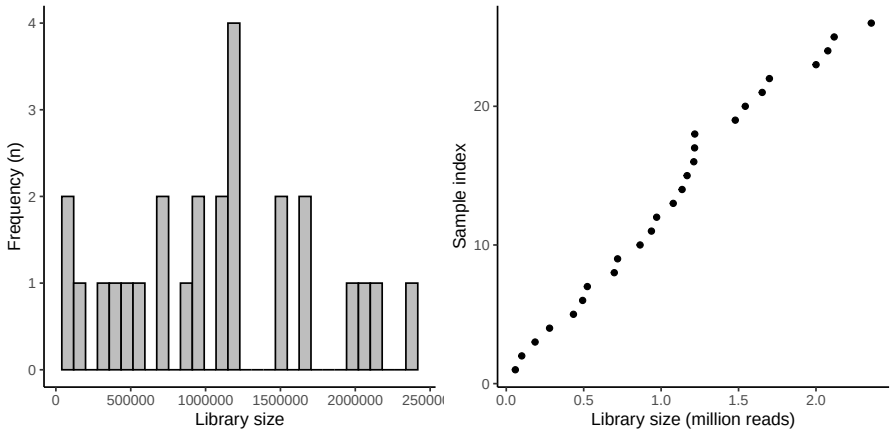


Figure 5.3: Library size distribution.

```
          DataFrame
plotColData(tse,"sum","X.SampleID", colour_by = "SampleType") +
    theme(axis.text.x = element_text(angle = 45, hjust=1)) +
    labs(y = "Library size (N)", x = "Sample ID")
```

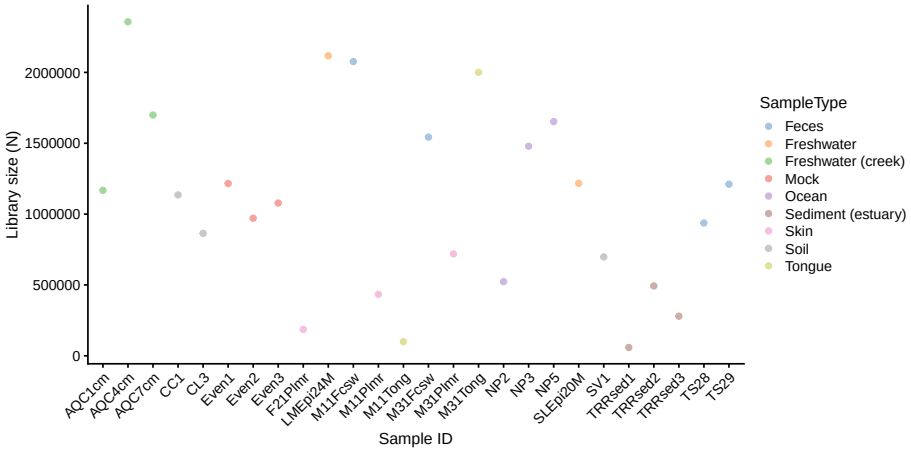


Figure 5.4: Library sizes per sample.

```
plotColData(tse,"sum","SampleType", colour_by = "SampleType") +
    theme(axis.text.x = element_text(angle = 45, hjust=1))
```

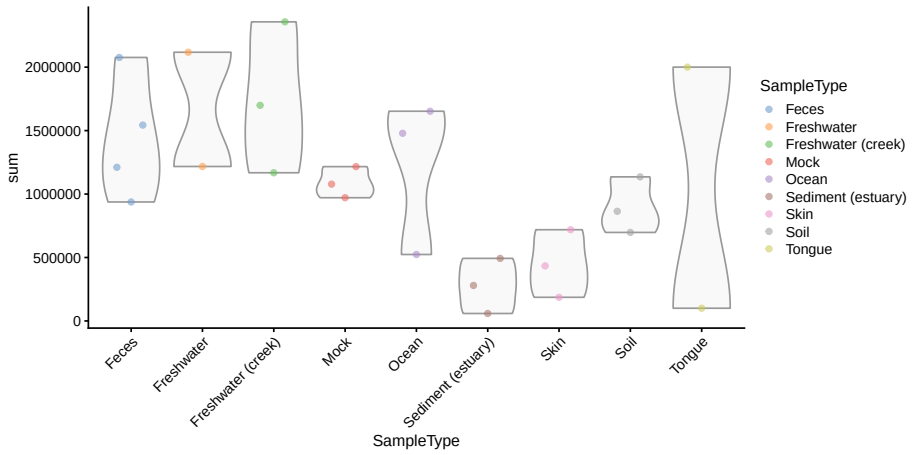


Figure 5.5: Library sizes per sample type.

In addition, data can be rarefied with subsampleCounts, which normalises the samples to an equal number of reads. However, this practice has been discouraged for the analysis of differentially abundant microorganisms (see (McMurdie and Holmes, 2014)).

5.3.3 Contaminant sequences

Samples might be contaminated with exogenous sequences. The impact of each contaminant can be estimated based on their frequencies and concentrations across the samples.

The following decontam functions are based on the (Davis et al., 2018) and support such functionality:

- `isContaminant`, `isNotContaminant`
- `addContaminantQC`, `addNotContaminantQC`

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS
```

```
Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] patchwork_1.1.2                dplyr_1.1.1
 [3] scater_1.26.1                  scuttle_1.8.4
 [5] miaViz_1.7.5                   ggraph_2.1.0
 [7] ggplot2_3.4.1                  miaTime_0.1.20
 [9] mia_1.7.11                     MultiAssayExperiment_1.24.0
[11] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
[13] XVector_0.38.0                 SingleCellExperiment_1.20.1
[15] SummarizedExperiment_1.28.0    Biobase_2.58.0
[17] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[19] IRanges_2.32.0                 S4Vectors_0.36.2
[21] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[23] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[25] rebook_1.6.0

loaded via a namespace (and not attached):
  [1] utf8_1.2.3                 tidyselect_1.2.0
  [3] RSQLite_2.3.0              AnnotationDbi_1.58.0
  [5] grid_4.2.1                 TSP_1.2-3
  [7] BiocParallel_1.32.6        Rtsne_0.16
  [9] munsell_0.5.0              ScaledMatrix_1.6.0
 [11] codetools_0.2-19           withr_2.5.0
 [13] colorspace_2.1-0           filelock_1.0.2
 [15] highr_0.10                 knitr_1.42
 [17] ca_0.71.1                  labeling_0.4.2
 [19] GenomeInfoDbData_1.2.9     polyclip_1.10-4
 [21] bit64_4.0.5                farver_2.1.1
 [23] vctrs_0.6.1                treeio_1.22.0
 [25] generics_0.1.3             xfun_0.38
 [27] R6_2.5.1                   doParallel_1.0.17
```

```
 [29] ggbeeswarm_0.7.1            clue_0.3-64
 [31] graphlayouts_0.8.4          rsvd_1.0.5
 [33] seriation_1.4.2             locfit_1.5-9.7
 [35] bitops_1.0-7                cachem_1.0.7
 [37] gridGraphics_0.5-1          DelayedArray_0.24.0
 [39] scales_1.2.1                SEtools_1.10.0
 [41] beeswarm_0.4.0              gtable_0.3.3
 [43] beachmat_2.14.0             sva_3.44.0
 [45] tidygraph_1.2.3             rlang_1.1.0
 [47] genefilter_1.78.0           GlobalOptions_0.1.2
 [49] splines_4.2.1               lazyeval_0.2.2
 [51] BiocManager_1.30.20         yaml_2.3.7
 [53] reshape2_1.4.4              tools_4.2.1
 [55] bookdown_0.33               ggplotify_0.1.0
 [57] decontam_1.18.0             RColorBrewer_1.1-3
 [59] Rcpp_1.0.10                 plyr_1.8.8
 [61] sparseMatrixStats_1.10.0    zlibbioc_1.44.0
 [63] purrr_1.0.1                 RCurl_1.98-1.12
 [65] GetoptLong_1.0.5            viridis_0.6.2
 [67] cowplot_1.1.1               ggrepel_0.9.3
 [69] cluster_2.1.4               DECIPHER_2.26.0
 [71] magrittr_2.0.3              data.table_1.14.8
 [73] openxlsx_4.2.5.2            circlize_0.4.15
 [75] ggnewscale_0.4.8            randomcoloR_1.1.0.1
 [77] evaluate_0.20               xtable_1.8-4
 [79] XML_3.99-0.14               gridExtra_2.3
 [81] shape_1.4.6                 compiler_4.2.1
 [83] tibble_3.2.1                V8_4.2.2
 [85] crayon_1.5.2                htmltools_0.5.5
 [87] ggfun_0.0.9                 mgcv_1.8-42
 [89] tidyr_1.3.0                 geneplotter_1.74.0
 [91] aplot_0.1.10                DBI_1.1.3
 [93] tweenr_2.0.2                ComplexHeatmap_2.12.1
 [95] MASS_7.3-58.3               Matrix_1.5-3
 [97] permute_0.9-7               cli_3.6.1
 [99] parallel_4.2.1              igraph_1.4.1
[101] pkgconfig_2.0.3             dir.expiry_1.4.0
[103] registry_0.5-1              foreach_1.5.2
[105] ggtree_3.4.4                annotate_1.74.0
[107] vipor_0.4.5                 DirichletMultinomial_1.40.0
[109] yulab.utils_0.0.6           stringr_1.5.0
[111] digest_0.6.31               vegan_2.6-4
[113] graph_1.74.0                rmarkdown_2.21
[115] tidytree_0.4.2              edgeR_3.38.4
[117] DelayedMatrixStats_1.20.0   curl_5.0.0
[119] rjson_0.2.21                lifecycle_1.0.3
```

```
[121] nlme_3.1-162              jsonlite_1.8.4
[123] BiocNeighbors_1.16.0      CodeDepends_0.6.5
[125] viridisLite_0.4.1         limma_3.52.4
[127] fansi_1.0.4               pillar_1.9.0
[129] lattice_0.20-45           KEGGREST_1.36.3
[131] fastmap_1.1.1             httr_1.4.5
[133] survival_3.5-5            glue_1.6.2
[135] zip_2.2.2                 sechm_1.4.1
[137] png_0.1-8                 iterators_1.0.14
[139] bit_4.0.5                 ggforce_0.4.1
[141] stringi_1.7.12            blob_1.2.4
[143] DESeq2_1.36.0             BiocSingular_1.14.0
[145] memoise_2.0.1             irlba_2.3.5.1
[147] ape_5.7-1
```

Chapter 6

Taxonomic Information

```
library(mia)
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
```

Taxonomic information is a key part of analyzing microbiome data and without it, any type of data analysis probably will not make much sense. However, the degree of detail of taxonomic information differs depending on the dataset and annotation data used.

Therefore, the mia package expects a loose assembly of taxonomic information and assumes certain key aspects:

- Taxonomic information is given as character vectors or factors in the `rowData` of a `SummarizedExperiment` object.
- The columns containing the taxonomic information must be named `domain`, `kingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species` or with a capital first letter.
- the columns must be given in the order shown above
- column can be omited, but the order must remain

6.1 Assigning taxonomic information.

There are a number of methods to assign taxonomic information. We like to give a short introduction about the methods available without ranking one over the other. This has to be your choice based on the result for the individual dataset.

6.1.1 dada2

The dada2 package (Callahan et al., 2016a) implements the `assignTaxonomy` function, which takes as input the ASV sequences associated with each row of data and a training dataset. For more information visit the dada2 homepage.

6.1.2 DECIPHER

The DECIPHER package (Wright, 2020) implements the `IDTAXA` algorithm to assign either taxonomic information or function information. For `mia` only the first option is of interest for now and more information can be found on the DECIPHER website.

6.2 Functions to access taxonomic information

`checkTaxonomy` checks whether the taxonomic information is usable for `mia`

```
checkTaxonomy(tse)
```

```
## [1] TRUE
```

Since the `rowData` can contain other data, `taxonomyRanks` will return the columns `mia` assumes to contain the taxonomic information.

```
taxonomyRanks(tse)
```

```
## [1] "Kingdom" "Phylum"  "Class"   "Order"   "Family"  "Genus"   "Species"
```

This can then be used to subset the `rowData` to columns needed.

```
rowData(tse)[,taxonomyRanks(tse)]
```

```
## DataFrame with 19216 rows and 7 columns
##            Kingdom        Phylum        Class        Order        Family
##        <character>   <character>  <character>  <character>   <character>
## 549322     Archaea Crenarchaeota Thermoprotei           NA            NA
## 522457     Archaea Crenarchaeota Thermoprotei           NA            NA
## 951        Archaea Crenarchaeota Thermoprotei Sulfolobales Sulfolobaceae
## 244423     Archaea Crenarchaeota        Sd-NA           NA            NA
## 586076     Archaea Crenarchaeota        Sd-NA           NA            NA
```

```
## ...            ...           ...           ...           ...           ...
## 278222    Bacteria           SR1            NA            NA            NA
## 463590    Bacteria           SR1            NA            NA            NA
## 535321    Bacteria           SR1            NA            NA            NA
## 200359    Bacteria           SR1            NA            NA            NA
## 271582    Bacteria           SR1            NA            NA            NA
##              Genus               Species
##        <character>            <character>
## 549322          NA                     NA
## 522457          NA                     NA
## 951     Sulfolobus Sulfolobusacidocalda..
## 244423          NA                     NA
## 586076          NA                     NA
## ...            ...                    ...
## 278222          NA                     NA
## 463590          NA                     NA
## 535321          NA                     NA
## 200359          NA                     NA
## 271582          NA                     NA
```

`taxonomyRankEmpty` checks for empty values in the given `rank` and returns a logical vector of `length(x)`.

```
all(!taxonomyRankEmpty(tse, rank = "Kingdom"))
```

```
## [1] TRUE
```

```
table(taxonomyRankEmpty(tse, rank = "Genus"))
```

```
##
## FALSE  TRUE
##  8008 11208
```

```
table(taxonomyRankEmpty(tse, rank = "Species"))
```

```
##
## FALSE  TRUE
##  1413 17803
```

`getTaxonomyLabels` is a multi-purpose function, which turns taxonomic information into a character vector of `length(x)`

```
head(getTaxonomyLabels(tse))
```

```
## [1] "Class:Thermoprotei"               "Class:Thermoprotei_1"
## [3] "Species:Sulfolobusacidocaldarius" "Class:Sd-NA"
## [5] "Class:Sd-NA_1"                    "Class:Sd-NA_2"
```

By default, this will use the lowest non-empty information to construct a string with the following scheme `level:value`. If all levels are the same, this part is omitted, but can be added by setting `with_rank = TRUE`.

```
phylum <- !is.na(rowData(tse)$Phylum) &
    ↪ vapply(data.frame(apply(rowData(tse)[,taxonomyRanks(tse)[3:7]],1L,is.na)),all,l
head(getTaxonomyLabels(tse[phylum,]))
```

```
## [1] "Crenarchaeota"    "Crenarchaeota_1"  "Crenarchaeota_2"  "Actinobacteria"
## [5] "Actinobacteria_1" "Spirochaetes"
```

```
head(getTaxonomyLabels(tse[phylum,], with_rank = TRUE))
```

```
## [1] "Phylum:Crenarchaeota"    "Phylum:Crenarchaeota_1"
## [3] "Phylum:Crenarchaeota_2"  "Phylum:Actinobacteria"
## [5] "Phylum:Actinobacteria_1" "Phylum:Spirochaetes"
```

By default the return value of `getTaxonomyLabels` contains only unique elements by passing it through `make.unique`. This step can be omitted by setting `make_unique = FALSE`.

```
head(getTaxonomyLabels(tse[phylum,], with_rank = TRUE,
 ↪ make_unique = FALSE))
```

```
## [1] "Phylum:Crenarchaeota"  "Phylum:Crenarchaeota"  "Phylum:Crenarchaeota"
## [4] "Phylum:Actinobacteria" "Phylum:Actinobacteria" "Phylum:Spirochaetes"
```

To apply the loop resolving function `resolveLoop` from the `TreeSummarizedExperiment` package (Huang, 2020) within `getTaxonomyLabels`, set `resolve_loops = TRUE`.

The function `getUniqueTaxa` gives a list of unique taxa for the specified taxonomic rank.

```
head(getUniqueTaxa(tse, rank = "Phylum"))
```

```
## [1] "Crenarchaeota"  "Euryarchaeota"  "Actinobacteria" "Spirochaetes"
## [5] "MVP-15"         "Proteobacteria"
```

6.2.1 Generate a taxonomic tree on the fly

To create a taxonomic tree, `taxonomyTree` used the information and returns a `phylo` object. Duplicate information from the `rowData` is removed.

```
taxonomyTree(tse)
```

```
##
## Phylogenetic tree with 1645 tips and 1089 internal nodes.
##
## Tip labels:
##   Species:Cenarchaeumsymbiosum, Species:pIVWA5, Species:CandidatusNitrososphaeragargensis, Spe
## Node labels:
##   root:ALL, Kingdom:Archaea, Phylum:Crenarchaeota, Class:C2, Class:Sd-NA, Class:Thaumarchaeota
##
## Rooted; includes branch lengths.
```

```
tse <- addTaxonomyTree(tse)
tse
```

```
## class: TreeSummarizedExperiment
## dim: 19216 26
## metadata(0):
## assays(1): counts
## rownames(19216): Class:Thermoprotei Class:Thermoprotei ... Phylum:SR1
##   Phylum:SR1
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (19216 rows)
## rowTree: 1 phylo tree(s) (1645 leaves)
## colLinks: NULL
## colTree: NULL
```

The implementation is based on the `toTree` function from the `TreeSummarizedExperiment` package (Huang, 2020).

6.3 Data agglomeration

One of the main applications of taxonomic information in regards to count data is to agglomerate count data on taxonomic levels and track the influence of changing conditions through these levels. For this `mia` contains the `agglomerateByRank` function. The ideal location to store the agglomerated data is as an alternative experiment.

```
tse <- relAbundanceCounts(tse)
altExp(tse, "Family") <- agglomerateByRank(tse, rank = "Family",
                                           agglomerateTree =
                                            ↪ TRUE)
altExp(tse, "Family")
```

```
## class: TreeSummarizedExperiment
## dim: 603 26
## metadata(1): agglomerated_by_rank
## assays(2): counts relabundance
## rownames(603): Class:Thermoprotei Family:Sulfolobaceae ...
##   Family:Thermodesulfobiaceae Phylum:SR1
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (603 rows)
## rowTree: 1 phylo tree(s) (496 leaves)
## colLinks: NULL
## colTree: NULL
```

If multiple assays (counts and relabundance) exist, both will be agglomerated.

```
assayNames(tse)
```

```
## [1] "counts"       "relabundance"
```

```
assayNames(altExp(tse, "Family"))
```

```
## [1] "counts"       "relabundance"
```

```
assay(altExp(tse, "Family"), "relabundance")[1:5,1:7]
```

```
##                            CL3       CC1 SV1 M31Fcsw M11Fcsw M31Plmr   M11Plmr
## Class:Thermoprotei   0.0000000 0.000e+00   0       0       0       0 0.000e+00
## Family:Sulfolobaceae 0.0000000 0.000e+00   0       0       0       0 2.305e-06
## Class:Sd-NA          0.0000000 0.000e+00   0       0       0       0 0.000e+00
## Order:NRP-J          0.0001991 2.070e-04   0       0       0       0 6.914e-06
## Family:SAGMA-X       0.0000000 6.165e-06   0       0       0       0 0.000e+00
```

```
assay(altExp(tse, "Family"), "counts")[1:5,1:7]
```

```
##                      CL3 CC1 SV1 M31Fcsw M11Fcsw M31Plmr M11Plmr
## Class:Thermoprotei     0   0   0       0       0       0       0
## Family:Sulfolobaceae   0   0   0       0       0       0       1
## Class:Sd-NA            0   0   0       0       0       0       0
## Order:NRP-J          172 235   0       0       0       0       3
## Family:SAGMA-X         0   7   0       0       0       0       0
```

`altExpNames` now consists of `Family` level data. This can be extended to use any taxonomic level listed in `mia::taxonomyRanks(tse)`.

6.4 Data transformation

Data transformations are common in microbiome analysis. Examples include the logarithmic transformation, calculation of relative abundances (percentages), and compositionality-aware transformations such as the centered log-ratio transformation (clr).

In mia package, transformations are applied to abundance data. The transformed abundance table is stored back to 'assays'. mia includes transformation function ('transformCounts()') which applies sample-wise or column-wise transformation when MARGIN = 'samples', feature-wise or row-wise transformation when MARGIN = 'features'.

For a complete list of available transformations and parameters, see function help.

```
assay(tse, "pseudo") <- assay(tse, "counts") + 1
tse <- transformCounts(tse, assay_name = "pseudo", method =
 ↪  "relabundance")
tse <- transformCounts(x = tse, assay_name = "relabundance",
 ↪  method = "clr",
                       pseudocount = 1, name =
                        ↪  "clr_transformation")
```

```
head(assay(tse, "clr_transformation"))
```

```
##                                         CL3        CC1        SV1    M31Fcsw
## Class:Thermoprotei                -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
## Class:Thermoprotei                -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
## Species:Sulfolobusacidocaldarius -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
## Class:Sd-NA                       -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
## Class:Sd-NA                       -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
## Class:Sd-NA                       -5.078e-05 -5.105e-05 -5.055e-05 -4.975e-05
##                                      M11Fcsw    M31Plmr    M11Plmr    F21Plmr
## Class:Thermoprotei                -4.947e-05 -4.931e-05 -4.879e-05 -4.671e-05
## Class:Thermoprotei                -4.947e-05 -4.931e-05 -4.879e-05 -4.671e-05
## Species:Sulfolobusacidocaldarius -4.947e-05 -4.931e-05 -4.658e-05 -4.671e-05
## Class:Sd-NA                       -4.947e-05 -4.931e-05 -4.879e-05 -4.671e-05
## Class:Sd-NA                       -4.947e-05 -4.931e-05 -4.879e-05 -4.671e-05
## Class:Sd-NA                       -4.947e-05 -4.931e-05 -4.879e-05 -4.671e-05
##                                      M31Tong    M11Tong   LMEpi24M   SLEpi20M
## Class:Thermoprotei                -4.846e-05 -4.257e-05 -4.756e-05 -4.837e-05
## Class:Thermoprotei                -4.846e-05 -4.257e-05 -4.756e-05 -4.918e-05
## Species:Sulfolobusacidocaldarius -4.846e-05 -4.257e-05 -4.756e-05 -4.918e-05
## Class:Sd-NA                       -4.846e-05 -4.257e-05 -4.756e-05 -4.918e-05
## Class:Sd-NA                       -4.846e-05 -4.257e-05 -4.756e-05 -4.918e-05
## Class:Sd-NA                       -4.846e-05 -4.257e-05 -4.756e-05 -4.918e-05
##                                       AQC1cm     AQC4cm     AQC7cm        NP2
## Class:Thermoprotei                -2.385e-05 -4.438e-06  2.787e-05 -4.731e-05
## Class:Thermoprotei                -4.660e-05 -4.568e-05 -4.428e-05 -4.915e-05
## Species:Sulfolobusacidocaldarius -4.660e-05 -4.652e-05 -4.777e-05 -4.915e-05
## Class:Sd-NA                       -4.660e-05 -3.726e-05 -3.090e-05 -4.915e-05
## Class:Sd-NA                       -4.660e-05 -4.568e-05 -4.719e-05 -4.915e-05
## Class:Sd-NA                       -4.660e-05 -4.610e-05 -4.603e-05 -4.915e-05
##                                         NP3        NP5    TRRsed1    TRRsed2
## Class:Thermoprotei                -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
## Class:Thermoprotei                -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
## Species:Sulfolobusacidocaldarius -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
## Class:Sd-NA                       -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
## Class:Sd-NA                       -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
## Class:Sd-NA                       -5.068e-05 -5.083e-05 -3.909e-05 -4.927e-05
##                                      TRRsed3       TS28       TS29      Even1
## Class:Thermoprotei                -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
## Class:Thermoprotei                -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
## Species:Sulfolobusacidocaldarius -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
## Class:Sd-NA                       -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
## Class:Sd-NA                       -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
## Class:Sd-NA                       -4.829e-05 -5.016e-05 -4.934e-05 -5.046e-05
```

```
##                                       Even2      Even3
## Class:Thermoprotei                -5.017e-05 -5.034e-05
## Class:Thermoprotei                -5.017e-05 -5.034e-05
## Species:Sulfolobusacidocaldarius -5.017e-05 -5.034e-05
## Class:Sd-NA                       -5.017e-05 -5.034e-05
## Class:Sd-NA                       -5.017e-05 -5.034e-05
## Class:Sd-NA                       -5.017e-05 -5.034e-05
```

- In 'pa' transformation, abundance table is converted to present/absent table.

```
tse <- transformCounts(tse, method = "pa")

head(assay(tse, "pa"))
```

```
##                                  CL3 CC1 SV1 M31Fcsw M11Fcsw M31Plmr M11Plmr
## Class:Thermoprotei                 0   0   0       0       0       0       0
## Class:Thermoprotei                 0   0   0       0       0       0       0
## Species:Sulfolobusacidocaldarius   0   0   0       0       0       0       1
## Class:Sd-NA                        0   0   0       0       0       0       0
## Class:Sd-NA                        0   0   0       0       0       0       0
## Class:Sd-NA                        0   0   0       0       0       0       0
##                                  F21Plmr M31Tong M11Tong LMEpi24M SLEpi20M
## Class:Thermoprotei                     0       0       0        0        1
## Class:Thermoprotei                     0       0       0        0        0
## Species:Sulfolobusacidocaldarius       0       0       0        0        0
## Class:Sd-NA                            0       0       0        0        0
## Class:Sd-NA                            0       0       0        0        0
## Class:Sd-NA                            0       0       0        0        0
##                                  AQC1cm AQC4cm AQC7cm NP2 NP3 NP5 TRRsed1
## Class:Thermoprotei                    1      1      1   1   0   0       0
## Class:Thermoprotei                    0      1      1   0   0   0       0
## Species:Sulfolobusacidocaldarius      0      0      0   0   0   0       0
## Class:Sd-NA                           0      1      1   0   0   0       0
## Class:Sd-NA                           0      1      1   0   0   0       0
## Class:Sd-NA                           0      1      1   0   0   0       0
##                                  TRRsed2 TRRsed3 TS28 TS29 Even1 Even2 Even3
## Class:Thermoprotei                     0       0    0    0     0     0     0
## Class:Thermoprotei                     0       0    0    0     0     0     0
## Species:Sulfolobusacidocaldarius       0       0    0    0     0     0     0
## Class:Sd-NA                            0       0    0    0     0     0     0
## Class:Sd-NA                            0       0    0    0     0     0     0
## Class:Sd-NA                            0       0    0    0     0     0     0
```

```
# list of abundance tables that assays slot contains
assays(tse)
```

```
## List of length 5
## names(5): counts relabundance pseudo clr_transformation pa
```

6.5 Pick specific

Retrieving of specific elements that are required for specific analysis. For instance, extracting abundances for a specific taxa in all samples or all taxa in one sample.

6.5.1 Abundances of all taxa in specific sample

```
taxa.abund.cc1 <- getAbundanceSample(tse,
                                     sample_id = "CC1",
                                     assay_name = "counts")
taxa.abund.cc1[1:10]
```

```
##               Class:Thermoprotei               Class:Thermoprotei
##                                0                                0
## Species:Sulfolobusacidocaldarius                      Class:Sd-NA
##                                0                                0
##                       Class:Sd-NA                      Class:Sd-NA
##                                0                                0
##                       Order:NRP-J                      Order:NRP-J
##                                1                                0
##                       Order:NRP-J                      Order:NRP-J
##                               194                                5
```

6.5.2 Abundances of specific taxa in all samples

```
taxa.abundances <- getAbundanceFeature(tse,
                                       feature_id =
                                        ↪ "Phylum:Bacteroidetes",
                                       assay_name = "counts")
taxa.abundances[1:10]
```

```
##     CL3     CC1     SV1 M31Fcsw M11Fcsw M31Plmr M11Plmr F21Plmr M31Tong M11Tong
##       2      18       2       0       0       0       0       1       0       0
```

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] mia_1.7.11                      MultiAssayExperiment_1.24.0
 [3] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
 [5] XVector_0.38.0                  SingleCellExperiment_1.20.1
 [7] SummarizedExperiment_1.28.0    Biobase_2.58.0
 [9] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[11] IRanges_2.32.0                 S4Vectors_0.36.2
[13] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[15] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[17] rebook_1.6.0

loaded via a namespace (and not attached):
 [1] nlme_3.1-162                bitops_1.0-7
 [3] DirichletMultinomial_1.40.0 bit64_4.0.5
 [5] filelock_1.0.2              tools_4.2.1
 [7] vegan_2.6-4                 utf8_1.2.3
 [9] R6_2.5.1                    irlba_2.3.5.1
[11] vipor_0.4.5                 mgcv_1.8-42
[13] DBI_1.1.3                   lazyeval_0.2.2
[15] colorspace_2.1-0            permute_0.9-7
[17] withr_2.5.0                 tidyselect_1.2.0
```

```
[19] gridExtra_2.3              bit_4.0.5
[21] compiler_4.2.1             graph_1.74.0
[23] cli_3.6.1                  BiocNeighbors_1.16.0
[25] DelayedArray_0.24.0        bookdown_0.33
[27] scales_1.2.1               stringr_1.5.0
[29] digest_0.6.31              yulab.utils_0.0.6
[31] rmarkdown_2.21             scater_1.26.1
[33] pkgconfig_2.0.3            htmltools_0.5.5
[35] decontam_1.18.0            sparseMatrixStats_1.10.0
[37] fastmap_1.1.1              rlang_1.1.0
[39] RSQLite_2.3.0              DelayedMatrixStats_1.20.0
[41] generics_0.1.3             jsonlite_1.8.4
[43] BiocParallel_1.32.6        dplyr_1.1.1
[45] RCurl_1.98-1.12            magrittr_2.0.3
[47] BiocSingular_1.14.0        GenomeInfoDbData_1.2.9
[49] scuttle_1.8.4              Matrix_1.5-3
[51] Rcpp_1.0.10                ggbeeswarm_0.7.1
[53] munsell_0.5.0              fansi_1.0.4
[55] DECIPHER_2.26.0            ape_5.7-1
[57] viridis_0.6.2              lifecycle_1.0.3
[59] stringi_1.7.12             yaml_2.3.7
[61] MASS_7.3-58.3              zlibbioc_1.44.0
[63] plyr_1.8.8                 blob_1.2.4
[65] grid_4.2.1                 parallel_4.2.1
[67] ggrepel_0.9.3              crayon_1.5.2
[69] dir.expiry_1.4.0           lattice_0.20-45
[71] splines_4.2.1              beachmat_2.14.0
[73] CodeDepends_0.6.5          knitr_1.42
[75] pillar_1.9.0               reshape2_1.4.4
[77] codetools_0.2-19           ScaledMatrix_1.6.0
[79] XML_3.99-0.14              glue_1.6.2
[81] evaluate_0.20              BiocManager_1.30.20
[83] vctrs_0.6.1                treeio_1.22.0
[85] gtable_0.3.3               purrr_1.0.1
[87] tidyr_1.3.0                cachem_1.0.7
[89] ggplot2_3.4.1              xfun_0.38
[91] rsvd_1.0.5                 tidytree_0.4.2
[93] viridisLite_0.4.1          tibble_3.2.1
[95] memoise_2.0.1              beeswarm_0.4.0
[97] cluster_2.1.4
```

Chapter 7

Community diversity

Diversity estimates are a central topic in microbiome data analysis.

There are three commonly employed levels of diversity measurements, which are trying to put a number on different aspects of the questions associated with diversity (Whittaker, 1960).

Many different ways for estimating such diversity measurements have been described in the literature. Which measurement is best or applicable for your samples, is not the aim of the following sections.

```
library(mia)
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
```

Alpha diversity, also sometimes interchangeably used with the term ***species diversity***, summarizes the distribution of species abundances in a given sample into a single number that depends on species richness and evenness. Diversity indices measure the overall community heterogeneity. A number of ecological diversity measures are available. The Hill coefficient combines many standard indices into a single equation that provides observed richness, inverse Simpson, and Shannon diversity, and generalized diversity as special cases. In general, diversity increases together with increasing richness and evenness. Sometimes richness, phylogenetic diversity, evenness, dominance, and rarity are considered to be variants of alpha diversity.

Richness refers to the total number of species in a community (sample). The simplest richness index is the number of observed species (observed richness). Assuming limited sampling from the community, however, this may underestimate the true species richness. Several estimators are available, including for instance ACE (A and SM, 1992) and Chao1 (A, 1984). Richness estimates are unaffected by species abundances.

Phylogenetic diversity was first proposed by (Faith, 1992). Unlike the diversity measures mentioned above, Phylogenetic diversity (PD) measure incorporates information from phylogenetic relationships stored in `phylo` tree between species in a community (sample). The Faith's PD is calculated as the sum of branch length of all species in a community (sample).

Evenness focuses on species abundances, and can thus complement the number of species. A typical evenness index is the Pielou's evenness, which is Shannon diversity normalized by the observed richness.

Dominance indices are in general negatively correlated with diversity, and sometimes used in ecological literature. High dominance is obtained when one or few species have a high share of the total species abundance in the community.

Rarity indices characterize the concentration of taxa at low abundance. Prevalence and detection thresholds determine rare taxa whose total concentration is represented as a rarity index.

7.1 Estimation

Alpha diversity can be estimated with wrapper functions that interact with other packages implementing the calculation, such as *vegan* (Oksanen et al., 2020).

7.1.1 Richness

Richness gives the number of features present within a community and can be calculated with `estimateRichness`. Each of the estimate diversity/richness/evenness/dominance functions adds the calculated measure(s) to the `colData` of the `SummarizedExperiment` under the given column `name`. Here, we calculate `observed` features as a measure of richness.

```
tse <- mia::estimateRichness(tse,
                             assay_name = "counts",
                             index = "observed",
                             name="observed")

head(colData(tse)$observed)
```

```
##     CL3     CC1     SV1 M31Fcsw M11Fcsw M31Plmr
##    6964    7679    5729    2667    2574    3214
```

This allows access to the values to be analyzed directly from the `colData`, for example by plotting them using `plotColData` from the *scater* package (McCarthy et al., 2020).

```
library(scater)
plotColData(tse,
            "observed",
            "SampleType",
            colour_by = "Final_Barcode") +
    theme(axis.text.x = element_text(angle=45,hjust=1)) +
  ylab(expression(Richness[Observed]))
```

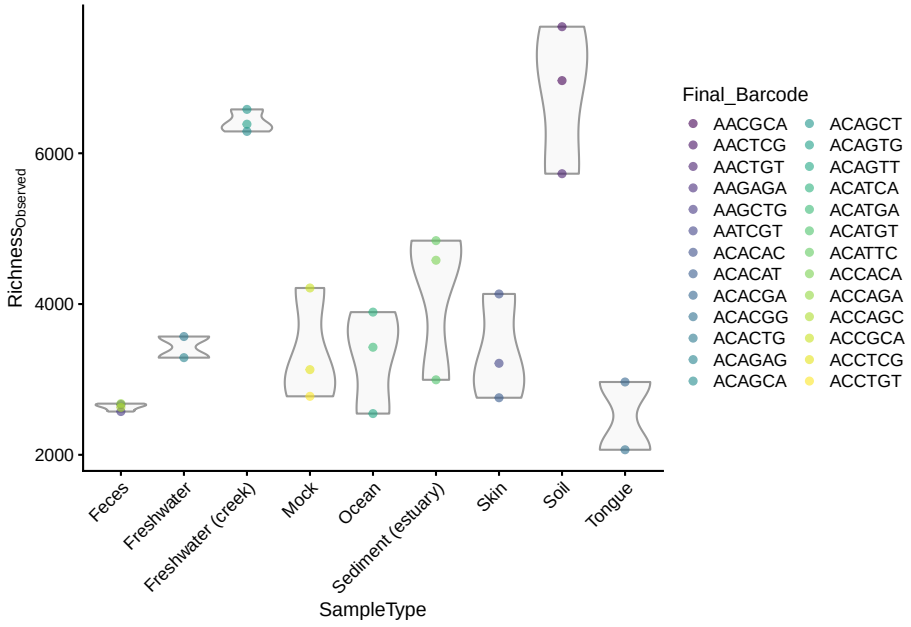


Figure 7.1: Shannon diversity estimates plotted grouped by sample type with colour-labeled barcode.

7.1.2 Diversity

The main function, `estimateDiversity`, calculates the selected diversity index based on the selected assay data.

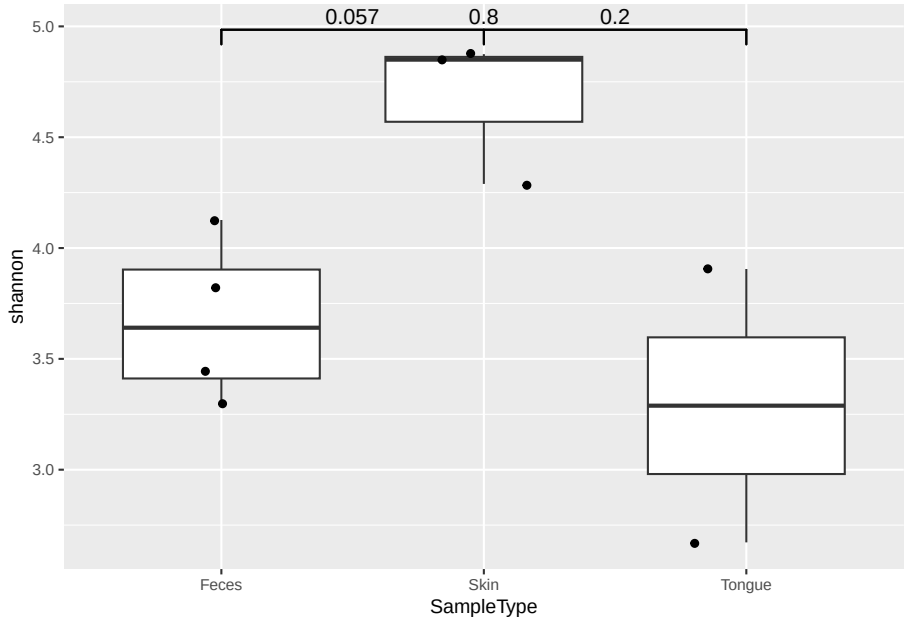
```
tse <- mia::estimateDiversity(tse,
                              assay_name = "counts",
                              index = "shannon",
                              name = "shannon")
head(colData(tse)$shannon)
```

```
##     CL3     CC1     SV1 M31Fcsw M11Fcsw M31Plmr
##   6.577   6.777   6.498   3.828   3.288   4.289
```

Alpha diversities can be visualized with boxplot. Here, Shannon index is compared between different sample type groups. Individual data points are visualized by plotting them as points with `geom_jitter`.

`geom_signif` is used to test whether these differences are statistically significant. It adds p-values to plot.

```
if( !require(ggsignif) ){
  install.packages("ggsignif")
}
library(ggplot2)
library(patchwork)
library(ggsignif)

# Subsets the data. Takes only those samples that are from feces,
 ↪  skin, or tongue,
# and creates data frame from the collected data
df <- as.data.frame(colData(tse)[colData(tse)$SampleType %in%
                                   c("Feces", "Skin", "Tongue"),
                                    ↪  ])

# Changes old levels with new levels
df$SampleType <- factor(df$SampleType)

# For significance testing, all different combinations are
 ↪  determined
comb <- split(t(combn(levels(df$SampleType), 2)),
           seq(nrow(t(combn(levels(df$SampleType), 2)))))

ggplot(df, aes(x = SampleType, y = shannon)) +
  # Outliers are removed, because otherwise each data point would
   ↪  be plotted twice;
  # as an outlier of boxplot and as a point of dotplot.
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0.2) +
  geom_signif(comparisons = comb, map_signif_level = FALSE) +
  theme(text = element_text(size = 10))
```



7.1.3 Faith phylogenetic diversity

The Faith index is returned by the function `estimateFaith`.

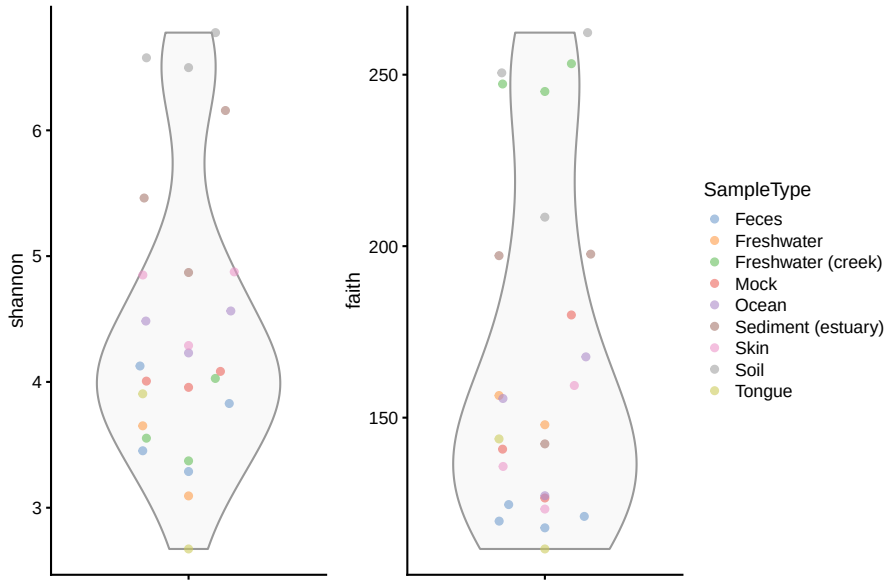
```
tse <- mia::estimateFaith(tse,
                          assay_name = "counts")
head(colData(tse)$faith)
```

```
## [1] 250.5 262.3 208.5 117.9 119.8 135.8
```

Note: because `tse` is a `TreeSummarizedExperiment` object, its phylogenetic tree is used by default. However, the optional argument `tree` must be provided if `tse` does not contain one.

Below a visual comparison between shannon and faith indices is shown with a violin plot.

```
plots <- lapply(c("shannon", "faith"),
                plotColData,
                object = tse, colour_by = "SampleType")
plots[[1]] + plots[[2]] +
  plot_layout(guides = "collect")
```



Alternatively, the phylogenetic diversity can be calculated by `mia::estimateDiversity`. This is a faster re-implementation of
the widely used function in *picante* (Kembel et al., 2010, W et al. (2010)).

Load `picante` R package and get the `phylo` stored in `rowTree`.

```
tse <- mia::estimateDiversity(tse,
                              assay_name = "counts",
                              index = "faith",
                              name = "faith")
```

7.1.4 Evenness

Evenness can be calculated with `estimateEvenness`.

```
tse <- estimateEvenness(tse,
                        assay_name = "counts",
                        index="simpson")
head(colData(tse)$simpson)
```

```
## [1] 0.026871 0.027197 0.047049 0.005179 0.004304 0.005011
```

7.1.5 Dominance

Dominance can be calculated with `estimateDominance`. Here, the `Relative index` is calculated which is the relative abundance of the most dominant species in the sample.

```
tse <- estimateDominance(tse,
                         assay_name = "counts",
                         index="relative")

head(colData(tse)$relative)
```

```
##     CL3     CC1     SV1 M31Fcsw M11Fcsw M31Plmr
## 0.03910 0.03226 0.01690 0.22981 0.21778 0.22329
```

7.1.6 Rarity

`mia` package provides one rarity index called log-modulo skewness. It can be calculated with `estimateDiversity`.

```
tse <- mia::estimateDiversity(tse,
                              assay_name = "counts",
                              index = "log_modulo_skewness")

head(colData(tse)$log_modulo_skewness)
```

```
## [1] 2.061 2.061 2.061 2.061 2.061 2.061
```

7.1.7 Divergence

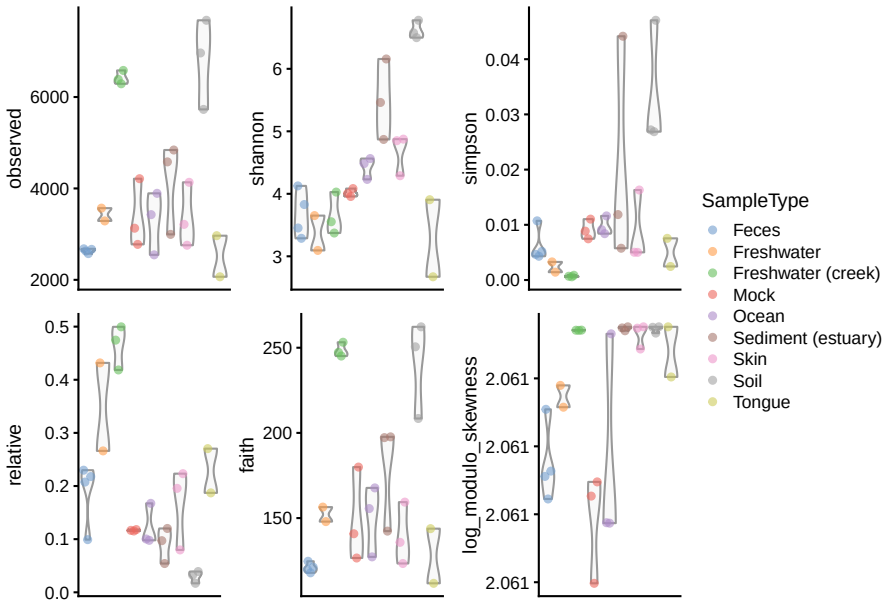
Divergence can be evaluated with `estimateDivergence`. Reference and algorithm for the calculation of divergence can be specified as `reference` and `FUN`, respectively.

```
tse <- mia::estimateDivergence(tse,
                               assay_name = "counts",
                               reference = "median",
                               FUN = vegan::vegdist)
```

7.2 Visualization

A plot comparing all the diversity measures calculated above and stored in `colData` can then be constructed directly.

```
plots <- lapply(c("observed", "shannon", "simpson", "relative",
 ↪  "faith", "log_modulo_skewness"),
                plotColData,
                object = tse,
                x = "SampleType",
                colour_by = "SampleType")

plots <- lapply(plots, "+",
                theme(axis.text.x = element_blank(),
                      axis.title.x = element_blank(),
                      axis.ticks.x = element_blank()))

((plots[[1]] | plots[[2]] | plots[[3]]) /
(plots[[4]] | plots[[5]] | plots[[6]])) +
  plot_layout(guides = "collect")
```



Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] patchwork_1.1.2               ggsignif_0.6.4
 [3] scater_1.26.1                 ggplot2_3.4.1
 [5] scuttle_1.8.4                 mia_1.7.11
 [7] MultiAssayExperiment_1.24.0   TreeSummarizedExperiment_2.1.4
 [9] Biostrings_2.66.0             XVector_0.38.0
[11] SingleCellExperiment_1.20.1   SummarizedExperiment_1.28.0
[13] Biobase_2.58.0                GenomicRanges_1.50.2
[15] GenomeInfoDb_1.34.9           IRanges_2.32.0
[17] S4Vectors_0.36.2              BiocGenerics_0.44.0
[19] MatrixGenerics_1.10.0         matrixStats_0.63.0-9003
[21] BiocStyle_2.24.0              rebook_1.6.0

loaded via a namespace (and not attached):
 [1] ggbeeswarm_0.7.1           colorspace_2.1-0
 [3] BiocNeighbors_1.16.0       farver_2.1.1
 [5] ggrepel_0.9.3              bit64_4.0.5
 [7] fansi_1.0.4                decontam_1.18.0
 [9] codetools_0.2-19           splines_4.2.1
[11] sparseMatrixStats_1.10.0   cachem_1.0.7
[13] knitr_1.42                 jsonlite_1.8.4
```

```
[15] cluster_2.1.4               graph_1.74.0
[17] BiocManager_1.30.20         compiler_4.2.1
[19] Matrix_1.5-3                fastmap_1.1.1
[21] lazyeval_0.2.2              cli_3.6.1
[23] BiocSingular_1.14.0         htmltools_0.5.5
[25] tools_4.2.1                 rsvd_1.0.5
[27] gtable_0.3.3                glue_1.6.2
[29] GenomeInfoDbData_1.2.9      reshape2_1.4.4
[31] dplyr_1.1.1                 Rcpp_1.0.10
[33] vctrs_0.6.1                 ape_5.7-1
[35] nlme_3.1-162                DECIPHER_2.26.0
[37] DelayedMatrixStats_1.20.0   xfun_0.38
[39] stringr_1.5.0               beachmat_2.14.0
[41] lifecycle_1.0.3             irlba_2.3.5.1
[43] XML_3.99-0.14               zlibbioc_1.44.0
[45] MASS_7.3-58.3               scales_1.2.1
[47] parallel_4.2.1              yaml_2.3.7
[49] memoise_2.0.1               gridExtra_2.3
[51] yulab.utils_0.0.6           stringi_1.7.12
[53] RSQLite_2.3.0               highr_0.10
[55] ScaledMatrix_1.6.0          tidytree_0.4.2
[57] permute_0.9-7               filelock_1.0.2
[59] BiocParallel_1.32.6         rlang_1.1.0
[61] pkgconfig_2.0.3             bitops_1.0-7
[63] evaluate_0.20               lattice_0.20-45
[65] purrr_1.0.1                 labeling_0.4.2
[67] treeio_1.22.0               CodeDepends_0.6.5
[69] cowplot_1.1.1               bit_4.0.5
[71] tidyselect_1.2.0            plyr_1.8.8
[73] magrittr_2.0.3              bookdown_0.33
[75] R6_2.5.1                    generics_0.1.3
[77] DelayedArray_0.24.0         DBI_1.1.3
[79] pillar_1.9.0                withr_2.5.0
[81] mgcv_1.8-42                 RCurl_1.98-1.12
[83] tibble_3.2.1                dir.expiry_1.4.0
[85] crayon_1.5.2                utf8_1.2.3
[87] rmarkdown_2.21              viridis_0.6.2
[89] grid_4.2.1                  blob_1.2.4
[91] vegan_2.6-4                 digest_0.6.31
[93] tidyr_1.3.0                 munsell_0.5.0
[95] DirichletMultinomial_1.40.0 beeswarm_0.4.0
[97] viridisLite_0.4.1           vipor_0.4.5
```

Chapter 8

Community similarity

Where alpha diversity focuses on community variation within a community (sample), beta diversity quantifies (dis-)similarites between communities (samples). Some of the most popular beta diversity measures in microbiome research include Bray-Curtis index (for compositional data), Jaccard index (for presence / absence data, ignoring abundance information), Aitchison distance (Euclidean distance for clr transformed abundances, aiming to avoid the compositionality bias), and the Unifrac distances (that take into account the phylogenetic tree information). Only some of the commonly used beta diversity measures are actual *distances*; this is a mathematically well-defined concept and many ecological beta diversity measures, such as Bray-Curtis index, are not proper distances. Therefore, the term dissimilarity or beta diversity is commonly used.

Technically, beta diversities are usually represented as `dist` objects, which contain triangular data describing the distance between each pair of samples. These distances can be further subjected to ordination. Ordination is a common concept in ecology that aims to reduce the dimensionality of the data for further evaluation or visualization. Ordination techniques aim to capture as much of essential information in the data as possible in a lower dimensional representation. Dimension reduction is bound to loose information but the common ordination techniques aim to preserve relevant information of sample similarities in an optimal way, which is defined in different ways by different methods. [TODO add references and/or link to ordination chapter instead?]

Some of the most common ordination methods in microbiome research include Principal Component Analysis (PCA), metric and non-metric multi-dimensional scaling (MDS, NMDS), The MDS methods are also known as Principal Coordinates Analysis (PCoA). Other recently popular techniques include t-SNE and UMAP.

8.1 Explained variance

The percentage of explained variance is typically shown for PCA ordination plots. This quantifies the proportion of overall variance in the data that is captured by the PCA axes, or how well the ordination axes reflect the original distances.

Sometimes a similar measure is shown for MDS/PCoA. The interpretation is generally different, however, and hence we do not recommend using it. PCA is a special case of PCoA with Euclidean distances. With non-Euclidean dissimilarities PCoA uses a trick where the pointwise dissimilarities are first cast into similarities in a Euclidean space (with some information loss i.e. stress) and then projected to the maximal variance axes. In this case, the maximal variance axes do not directly reflect the correspondence of the projected distances and original distances, as they do for PCA.

In typical use cases, we would like to know how well the ordination reflects the original similarity structures; then the quantity of interest is the so-called "stress" function, which measures the difference in pairwise similarities between the data points in the original (high-dimensional) vs. projected (low-dimensional) space.

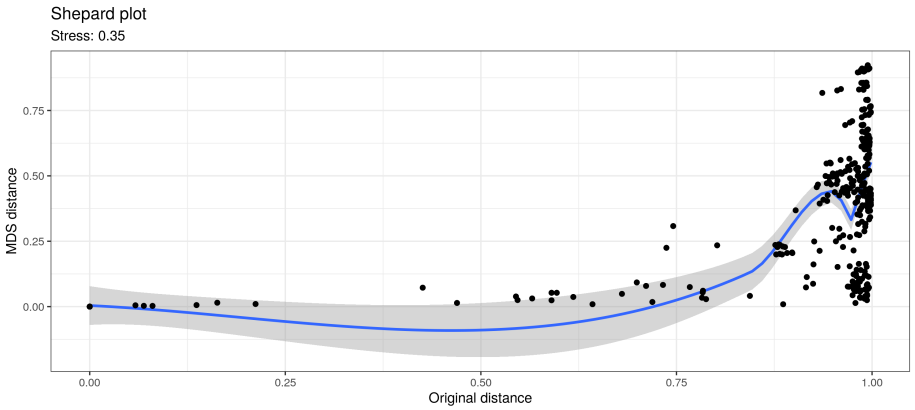
Hence, we propose that for PCoA and other ordination methods, users would report relative stress (varies in the unit interval; the smaller the better). This can be calculated as shown below. For further examples, check the note from Huber lab.

```
# Example data
library(mia)
data(GlobalPatterns, package="mia")

# Data matrix (features x samples)
tse <- GlobalPatterns
tse <- transformCounts(tse, method = "relabundance")

# Add group information Feces yes/no
colData(tse)$Group <- colData(tse)$SampleType=="Feces"

# Quantify dissimilarities in the original feature space
library(vegan)
x <- assay(tse, "relabundance") # Pick relabunance assay
 ↪  separately
d0 <- as.matrix(vegdist(t(x), "bray"))

# PCoA Ordination
pcoa <- as.data.frame(cmdscale(d0, k = 2))
names(pcoa) <- c("PCoA1", "PCoA2")
```

```
# Quantify dissimilarities in the ordination space
dp <- as.matrix(dist(pcoa))

# Calculate stress i.e. relative difference in the original and
# projected dissimilarities
stress <- sum((dp - d0)^2)/sum(d0^2)
```

Shepard plot visualizes the original versus projected (ordination) dissimilarities between the data points:

```
ord <- order(as.vector(d0))
df <- data.frame(d0 = as.vector(d0)[ord],
                 dmds = as.vector(dp)[ord])

library(ggplot2)
ggplot(aes(x = d0, y = dmds), data=df) +
       geom_smooth() +
       geom_point() +
       labs(title = "Shepard plot",
       x = "Original distance",
       y = "MDS distance",
            subtitle = paste("Stress:", round(stress, 2))) +
  theme_bw()
```



8.2 Community comparisons by beta diversity analysis

A typical comparison of community composition starts with a visual comparison of the groups on a 2D ordination.

Then we estimate relative abundances and MDS ordination based on Bray-Curtis (BC) dissimilarity between the groups, and visualize the results.

In the following examples dissimilarities are calculated by functions supplied to the `FUN` argument. This function can be defined by the user. It must return a `dist` function, which can then be used to calculate reduced dimensions either via ordination methods (such as MDS or NMDS), and the results can be stored in the `reducedDim`.

This entire process is wrapped in the `runMDS` and `runNMDS` functions.

```
library(scater)

# Bray-Curtis is usually applied to relative abundances
tse <- transformCounts(tse, method = "relabundance")
# Perform PCoA
tse <- runMDS(tse, FUN = vegan::vegdist, method = "bray", name =
 ↪  "PCoA_BC", exprs_values = "relabundance")
```

Sample similarities can be visualized on a lower-dimensional display (typically 2D) using the `plotReducedDim` function in the `scater` package. This provides also further tools to incorporate additional information using variations in color, shape or size. Are there visible differences between the groups?

```
# Create ggplot object
p <- plotReducedDim(tse, "PCoA_BC", colour_by = "Group")

# Add explained variance for each axis
e <- attr(reducedDim(tse, "PCoA_BC"), "eig");
rel_eig <- e/sum(e[e>0])
p <- p + labs(x = paste("PCoA 1 (", round(100 * rel_eig[[1]],1),
 ↪  "%", ")", sep = ""),
              y = paste("PCoA 2 (", round(100 * rel_eig[[2]],1),
               ↪  "%", ")", sep = ""))

print(p)
```

With additional tools from the `ggplot2` universe, comparisons can be performed informing on the applicability to visualize sample similarities in a meaningful way.

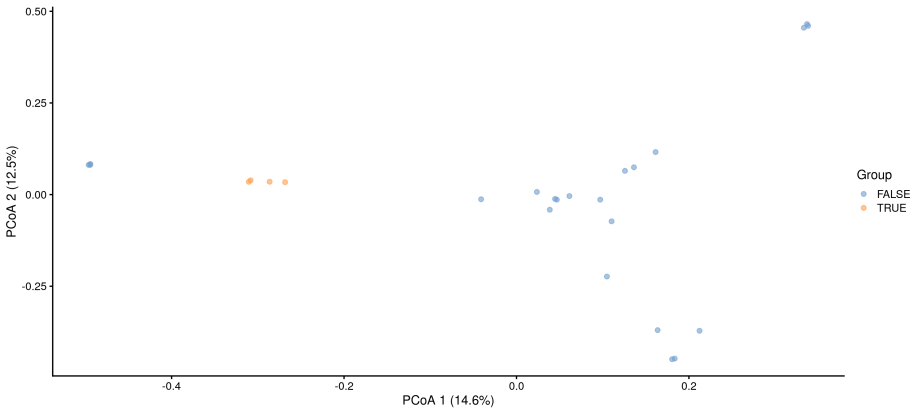


Figure 8.1: MDS plot based on the Bray-Curtis distances on the GlobalPattern dataset.

```
tse <- runMDS(tse, FUN = vegan::vegdist, name = "MDS_euclidean",
              method = "euclidean", exprs_values = "counts")
tse <- runNMDS(tse, FUN = vegan::vegdist, name = "NMDS_BC")
```

```
## initial  value 47.733208
## iter   5 value 33.853364
## iter  10 value 32.891200
## final  value 32.823570
## converged
```

```
tse <- runNMDS(tse, FUN = vegan::vegdist, name =
 ↪  "NMDS_euclidean",
               method = "euclidean")
```

```
## initial  value 31.882673
## final  value 31.882673
## converged
```

```
plots <- lapply(c("PCoA_BC", "MDS_euclidean", "NMDS_BC",
 ↪  "NMDS_euclidean"),
                plotReducedDim,
                object = tse,
                colour_by = "Group")

library(patchwork)
```

```
plots[[1]] + plots[[2]] + plots[[3]] + plots[[4]] +
  plot_layout(guides = "collect")
```

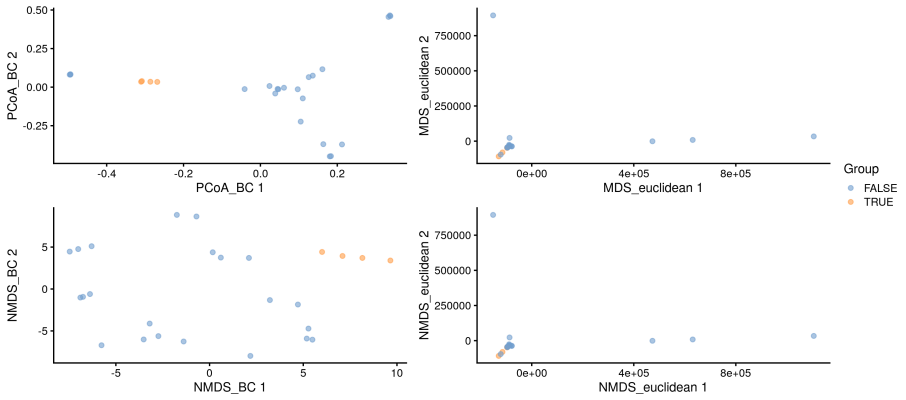


Figure 8.2: Comparison of MDS and NMDS plots based on the Bray-Curtis or euclidean distances on the GlobalPattern dataset.

The *Unifrac* method is a special case, as it requires data on the relationship of features in form on a `phylo` tree. `calculateUnifrac` performs the calculation to return a `dist` object, which can again be used within `runMDS`.

```
library(scater)
tse <- runMDS(tse, FUN = mia::calculateUnifrac, name = "Unifrac",
              tree = rowTree(tse),
              ntop = nrow(tse),
             exprs_values = "counts")
```

```
plotReducedDim(tse, "Unifrac", colour_by = "Group")
```

8.3 Other ordination methods

Other dimension reduction methods, such as `PCA`, `t-SNE` and `UMAP` are inherited directly from the `scater` package.

```
tse <- runPCA(tse, name = "PCA", exprs_values = "counts",
 ↪  ncomponents = 10)
```

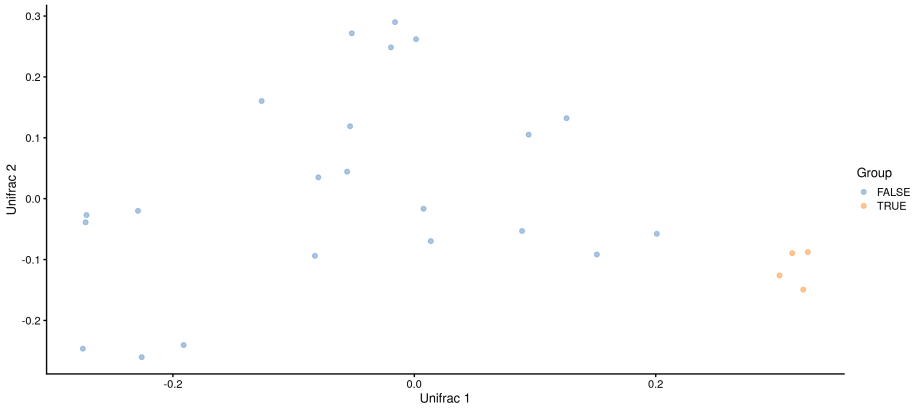


Figure 8.3: Unifrac distances scaled by MDS of the GlobalPattern dataset.

```
plotReducedDim(tse, "PCA", colour_by = "Group")
```

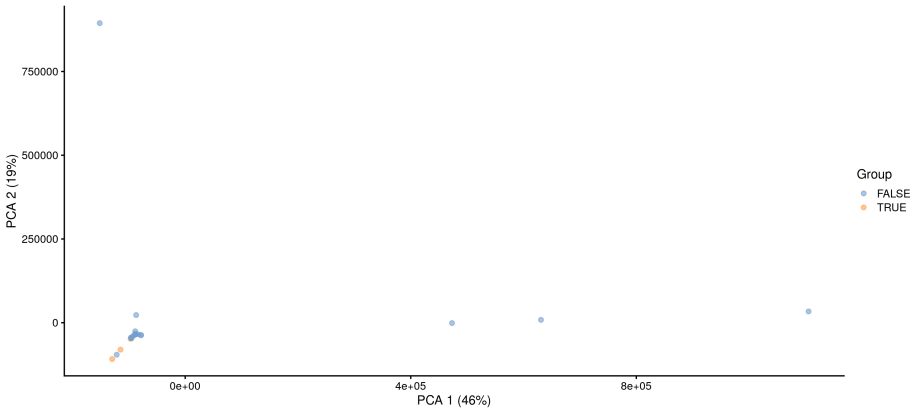


Figure 8.4: PCA plot on the GlobalPatterns data set containing sample from different sources.

As mentioned before, applicability of the different methods depends on your sample set.

FIXME: let us switch to UMAP for the examples?

```
tse <- runTSNE(tse, name = "TSNE", exprs_values = "counts",
 ↪  ncomponents = 3)
```

```
plotReducedDim(tse, "TSNE", colour_by = "Group", ncomponents =
 ↪  c(1:3))
```

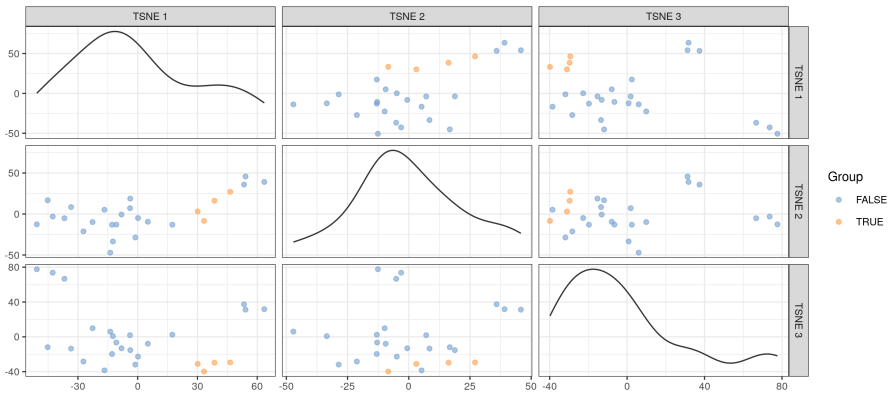


Figure 8.5: t-SNE plot on the GlobalPatterns data set containing sample from different sources.

As a final note, `mia` provides functions for the evaluation of additional dissimilarity indices, such as: * `calculateJSD`, `runJSD` (Jensen-Shannon divergence) * `calculateNMDS`, `plotNMDS` (non-metric multi-dimensional scaling) * `calculateCCA`, `runCCA` (Canonical Correspondence Analysis) * `calculateRDA`, `runRDA` (Redundancy Analysis) * `calculateOverlap`, `runOverlap` () * `calculateDPCoA`, `runDPCoA` (Double Principal Coordinate Analysis)

Redundancy analysis is similar to PCA, however, it takes into account covariates. It aims to maximize the variance in respect of covariates. The results shows how much each covariate affects.

```
# Load required packages
if(!require("vegan")){
    install.packages("vegan")
    library("vegan")
}
if(!require("stringr")){
    install.packages("stringr")
    library("stringr")
}
if(!require("knitr")){
    install.packages("knitr")
    library("knitr")
}
# Load data
```

```
data(enterotype)
# Covariates that are being analyzed
variable_names <- c("ClinicalStatus", "Gender", "Age")

# Apply relative transform
enterotype <- transformCounts(enterotype, method =
 ↪  "relabundance")

# Create a formula
formula <- as.formula(paste0("assay ~ ", str_c(variable_names,
 ↪  collapse = " + ")) )

# # Perform RDA
rda <- calculateRDA(enterotype, assay_name = "relabundance",
                    formula = formula, distance = "bray",
                     ↪  na.action = na.exclude)
# Get the rda object
rda <- attr(rda, "rda")
# Calculate p-value and variance for whole model
# Recommendation: use 999 permutations instead of 99
set.seed(436)
permanova <- anova.cca(rda, permutations = 99)
# Create a data.frame for results
rda_info <- as.data.frame(permanova)["Model", ]

# Calculate p-value and variance for each variable
# by = "margin" --> the order or variables does not matter
set.seed(4585)
permanova <- anova.cca(rda, by = "margin",  permutations = 99)
# Add results to data.frame
rda_info <- rbind(rda_info, permanova)

# Add info about total variance
rda_info[ , "Total variance"] <- rda_info["Model", 2] +
    rda_info["Residual", 2]

# Add info about explained variance
rda_info[ , "Explained variance"] <- rda_info[ , 2] /
    rda_info[ , "Total variance"]

# Loop through variables, calculate homogeneity
homogeneity <- list()
# Get colDtaa
coldata <- colData(enterotype)
# Get assay
```

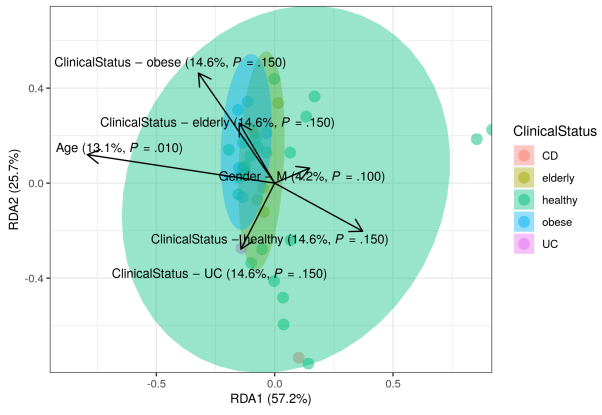
```
assay <- t(assay(enterotype, "relabundance"))
for( variable_name in rownames(rda_info) ){
    # If data is continuous or discrete
    if( variable_name %in% c("Model", "Residual") ||
        length(unique(coldata[[variable_name]])) /
        length(coldata[[variable_name]]) > 0.2 ){
        # Do not calculate homogeneity for continuous data
        temp <- NA
    } else{
        # Calculate homogeneity for discrete data
        # Calculate homogeneity
        set.seed(413)
        temp <- anova(
            betadisper(
                vegdist(assay, method = "bray"),
                group = coldata[[variable_name]] ),
            permutations = permutations )["Groups", "Pr(>F)"]
    }
    # Add info to the list
    homogeneity[[variable_name]] <- temp
}
# Add homogeneity to information
rda_info[["Homogeneity p-value (NULL hyp: distinct/homogeneous
 ↪  --> permanova suitable)"]] <-
    homogeneity

kable(rda_info)
```

	Df	SumOfSqs	F	Pr(>F)	Total variance	Explained variance	Homogene
Model	6	1.1157	1.940	0.05	3.991	0.2795	NA
ClinicalStatus	4	0.5837	1.522	0.15	3.991	0.1463	0.044277..
Gender	1	0.1679	1.751	0.10	3.991	0.0421	0.522999..
Age	1	0.5245	5.471	0.01	3.991	0.1314	0.000369..
Residual	30	2.8757	NA	NA	3.991	0.7205	NA

```
# Load ggord for plotting
if(!require("ggord")){
    if(!require("devtools")){
        install.packages("devtools")
        library("devtools")
    }
    install_github("https://github.com/fawda123/ggord/")
    library("ggord")
}
if(!require("ggplot2")){
```

```
    install.packages("ggplot2")
    library("ggplot2")
}
# Since na.exclude was used, if there were rows missing
 ↪  information, they were
# dropped off. Subset coldata so that it matches with rda.
coldata <- coldata[ rownames(rda$CCA$wa), ]

# Adjust names
# Get labels of vectors
vec_lab_old <- rownames(rda$CCA$biplot)

# Loop through vector labels
vec_lab <- sapply(vec_lab_old, FUN = function(name){
    # Get the variable name
    variable_name <- variable_names[ str_detect(name,
 ↪  variable_names) ]
    # If the vector label includes also group name
    if( !any(name %in% variable_names) ){
        # Get the group names
        group_name <- unique( coldata[[variable_name]] )[
        which( paste0(variable_name, unique(
         ↪  coldata[[variable_name]] )) == name ) ]
        # Modify vector so that group is separated from variable
         ↪  name
        new_name <- paste0(variable_name, " \U2012 ", group_name)
    } else{
        new_name <- name
    }
    # Add percentage how much this variable explains, and p-value
    new_name <- expr(paste(!!new_name, " (",
                           !!format(round(
                            ↪  rda_info[variable_name, "Explained
                            ↪  variance"]*100, 1), nsmall = 1),
                           "%, ",italic("P"), " = ",
                           !!gsub("0\\.","\\.", format(round(
                            ↪  rda_info[variable_name, "Pr(>F)"],
                            ↪  3),
                                                       nsmall =
                                                        ↪  3)),
                                                        ↪  ")"))

    return(new_name)
})
# Add names
```

```
names(vec_lab) <- vec_lab_old

# Create labels for axis
xlab <- paste0("RDA1 (", format(round(
 ↪  rda$CCA$eig[[1]]/rda$CCA$tot.chi*100, 1), nsmall = 1 ), "%)")
ylab <- paste0("RDA2 (", format(round(
 ↪  rda$CCA$eig[[2]]/rda$CCA$tot.chi*100, 1), nsmall = 1 ), "%)")

# Create a plot
plot <- ggord(rda, grp_in = coldata[["ClinicalStatus"]], vec_lab
 ↪  = vec_lab,
              alpha = 0.5,
              size = 4, addsize = -4,
              #ext= 0.7,
              txt = 3.5, repel = TRUE,
              #coord_fix = FALSE
          ) +
    # Adjust titles and labels
    guides(colour = guide_legend("ClinicalStatus"),
           fill = guide_legend("ClinicalStatus"),
           group = guide_legend("ClinicalStatus"),
           shape = guide_legend("ClinicalStatus"),
           x = guide_axis(xlab),
           y = guide_axis(ylab)) +
    theme( axis.title = element_text(size = 10) )
plot
```



From RDA plot, we can see that only age has significant affect on microbial profile.

8.4 Visualizing the most dominant genus on PCoA

In this section we visualize most dominant genus on PCoA. A similar visualization was proposed by Salosensaari et al. (2021).

Let us agglomerate the data at a Genus level and getting the dominant taxa per sample.

```
# Agglomerate to genus level
tse_Genus <- agglomerateByRank(tse, rank="Genus")
# Convert to relative abundances
tse_Genus <- transformCounts(tse, method = "relabundance",
 ↪  assay_name="counts")
# Add info on dominant genus per sample
tse_Genus <- addPerSampleDominantTaxa(tse_Genus,
 ↪  assay_name="relabundance", name = "dominant_taxa")
```

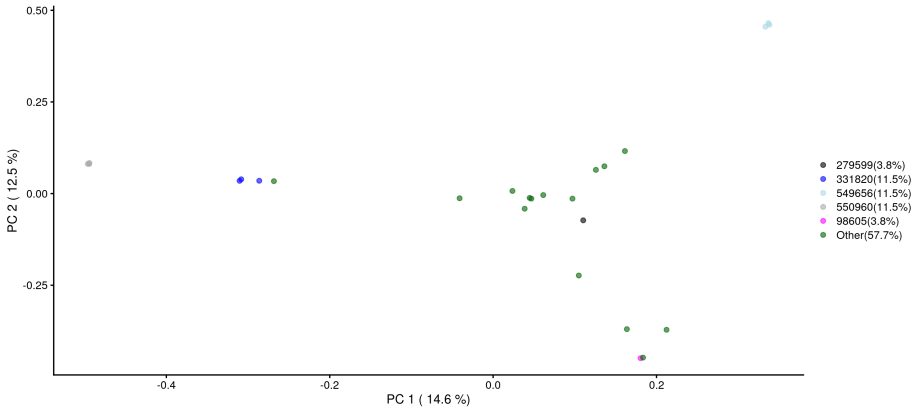
Performing PCoA with Bray-Curtis dissimilarity.

```
tse_Genus <- runMDS(tse_Genus, FUN = vegan::vegdist,
              name = "PCoA_BC", exprs_values = "relabundance")
```

Getting top taxa and visualizing the abundance on PCoA.

```
# Getting the top taxa
top_taxa <- getTopTaxa(tse_Genus,top = 6, assay_name =
 ↪  "relabundance")

# Naming all the rest of non top-taxa as "Other"
most_abundant <- lapply(colData(tse_Genus)$dominant_taxa,
                   function(x){if (x %in% top_taxa) {x} else
                    ↪  {"Other"}})

# Storing the previous results as a new column within colData
colData(tse_Genus)$most_abundant <- as.character(most_abundant)

# Calculating percentage of the most abundant
most_abundant_freq <- table(as.character(most_abundant))
most_abundant_percent <-
 ↪  round(most_abundant_freq/sum(most_abundant_freq)*100, 1)

# Retrieving the explained variance
e <- attr(reducedDim(tse_Genus, "PCoA_BC"), "eig");
```

```
var_explained <- e/sum(e[e>0])*100

# Visualization
plot <-plotReducedDim(tse_Genus,"PCoA_BC", colour_by =
 ↪  "most_abundant") +
  scale_colour_manual(values = c("black", "blue", "lightblue",
   ↪  "darkgray", "magenta", "darkgreen", "red"),

                       ↪  labels=paste0(names(most_abundant_percent),"(",most_abundant_
  labs(x=paste("PC 1 (",round(var_explained[1],1),"%)"),
       y=paste("PC 2 (",round(var_explained[2],1),"%)"),
       color="")
plot
```

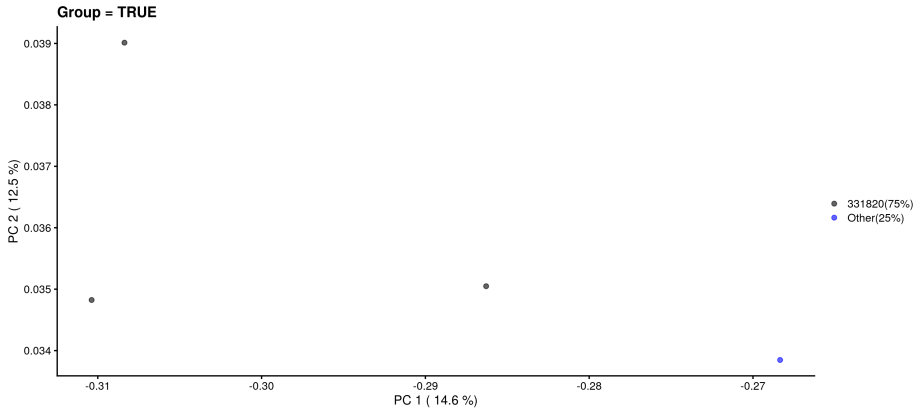


Note: A 3D interactive version of the earlier plot can be found from 17.

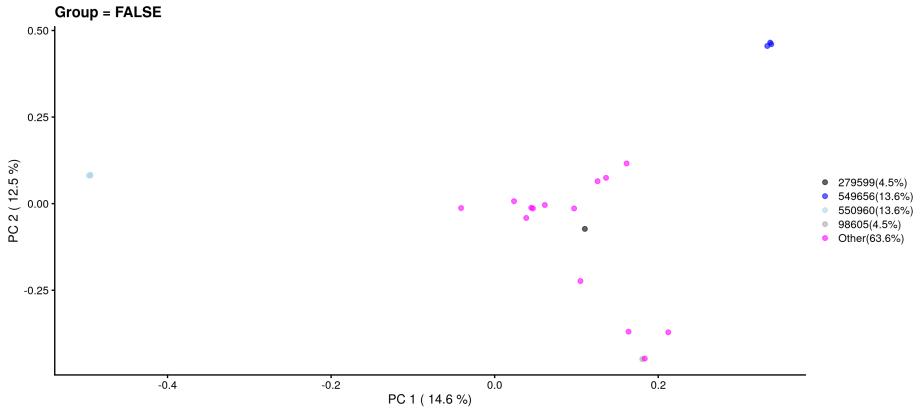
Similarly let's visualize and compare the sub-population.

```
# Calculating the frequencies and percentages for both categories
freq_TRUE <-
 ↪  table(as.character(most_abundant[colData(tse_Genus)$Group==TRUE]))
freq_FALSE <-
 ↪  table(as.character(most_abundant[colData(tse_Genus)$Group==FALSE]))
percent_TRUE <- round(freq_TRUE/sum(freq_TRUE)*100, 1)
percent_FALSE <- round(freq_FALSE/sum(freq_FALSE)*100, 1)

# Visualization
plotReducedDim(tse_Genus[,colData(tse_Genus)$Group==TRUE],
                          "PCoA_BC", colour_by = "most_abundant")
                           ↪  +
  scale_colour_manual(values = c("black", "blue", "lightblue",
   ↪  "darkgray", "magenta", "darkgreen", "red"),
```

```
                        ↪  labels=paste0(names(percent_TRUE),"(",percent_TRUE,"%)"))+
  labs(x=paste("PC 1 (",round(var_explained[1],1),"%)"),
       y=paste("PC 2 (",round(var_explained[2],1),"%)"),
       title = "Group = TRUE", color="")
```




```
plotReducedDim(tse_Genus[,colData(tse_Genus)$Group==FALSE],
                        "PCoA_BC", colour_by = "most_abundant")
                         ↪  +
  scale_colour_manual(values = c("black", "blue", "lightblue",
   ↪  "darkgray", "magenta", "darkgreen", "red"),

                        ↪  labels=paste0(names(percent_FALSE),"(",percent_FALSE,"%)"))+
  labs(x=paste("PC 1 (",round(var_explained[1],1),"%)"),
       y=paste("PC 2 (",round(var_explained[2],1),"%)"),
       title = "Group = FALSE", color="")
```



8.4.1 Testing differences in community composition between sample groups

The permutational analysis of variance (PERMANOVA) (Anderson, 2001) is a widely used non-parametric multivariate method that can be used to estimate the actual statistical significance of differences in the observed community composition between two groups of samples.

PERMANOVA evaluates the hypothesis that the centroids and dispersion of the community are equivalent between the compared groups. A small p-value indicates that the compared groups have, on average, a different community composition.

This method is implemented in the `vegan` package in the function `adonis2`.

Note:

It is recommended to `by = "margin"`. It specifies that each variable's marginal effect is analyzed individually.

When `by = "terms"` (the default) the order of variables matters; each variable is analyzed sequentially, and the result is different when more than 1 variable is introduced and their order is differs. (Check comparison)

We can perform PERMANOVA with `adonis2` function or by first performing distance-based redundancy analysis (dbRDA), and then applying permutational test for result of redundancy analysis. Advantage of the latter approach is that by doing so we can get coefficients: how much each taxa affect to the result.

```
if( !require(vegan) ){
    BiocManager::install("vegan")
    library("vegan")
}
# Agglomerate data to Species level
tse <- agglomerateByRank(tse, rank = "Species")

# Set seed for reproducibility
set.seed(1576)
# We choose 99 random permutations. Consider applying more (999
 ↪  or 9999) in your
# analysis.
permanova <- adonis2(t(assay(tse,"relabundance")) ~ Group,
                     by = "margin", # each term (here only
                      ↪  'Group') analyzed individually
                     data = colData(tse),
                     method = "euclidean",
                     permutations = 99)
```

```
# Set seed for reproducibility
set.seed(1576)
# Perform dbRDA
dbrda <- dbrda(t(assay(tse,"relabundance")) ~ Group,
               data = colData(tse))
# Perform permutational analysis
permanova2 <- anova.cca(dbrda,
                        by = "margin", # each term (here only
                         ↪  'Group') analyzed individually
                        method = "euclidean",
                        permutations = 99)

# Get p-values
p_values <- c( permanova["Group", "Pr(>F)"], permanova2["Group",
 ↪  "Pr(>F)"] )
p_values <-as.data.frame(p_values)
rownames(p_values) <- c("adonis2", "dbRDA+anova.cca")
p_values
```

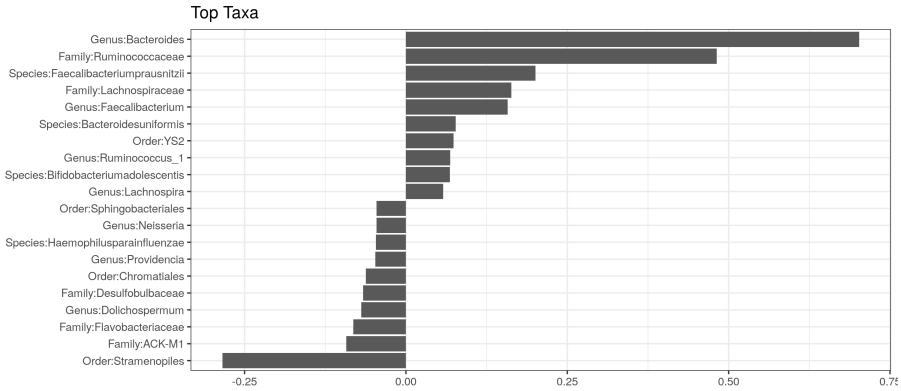
```
##                  p_values
## adonis2              0.02
## dbRDA+anova.cca      0.02
```

As we can see, the community composition is significantly different between the groups ($p < 0.05$), and these two methods give equal p-values.

Let us visualize the model coefficients for species that exhibit the largest differences between the groups. This gives some insights into how the groups tend to differ from each other in terms of community composition.

```
# Add taxa info
sppscores(dbrda) <- t(assay(tse,"relabundance"))
# Get coefficients
coef <- dbrda$CCA$v
# Get the taxa with biggest weights
top.coef <- head( coef[rev(order(abs(coef))), , drop = FALSE],
 ↪  20)
# Sort weights in increasing order
top.coef <- top.coef[ order(top.coef), ]
# Get top names
top_names <- names(top.coef)[ order(abs(top.coef), decreasing =
 ↪  TRUE) ]
```

```
ggplot(data.frame(x = top.coef,
                  y = factor(names(top.coef),
                                          unique(names(top.coef)))),
        aes(x = x, y = y)) +
    geom_bar(stat="identity") +
    labs(x="",y="",title="Top Taxa") +
    theme_bw()
```



In the above example, the largest differences between the two groups can be attributed to *Genus:Bacteroides* (elevated in the first group) and *Family:Ruminococcaceae* (elevated in the second group), and many other co-varying species.

8.4.2 Checking the homogeneity condition

It is important to note that the application of PERMANOVA assumes homogeneous group dispersions (variances). This can be tested with the PERMDISP2 method (Anderson, 2006) by using the same assay and distance method than in PERMANOVA.

```
anova( betadisper(vegdist(t(assay(tse, "counts"))),
 ↪  colData(tse)$Group) )
```

```
## Analysis of Variance Table
##
## Response: Distances
##           Df Sum Sq Mean Sq F value  Pr(>F)
## Groups     1 0.2385  0.2385     103 3.6e-10 ***
## Residuals 24 0.0554  0.0023
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If the groups have similar dispersion, PERMANOVA can be seen as an appropriate choice for comparing community compositions.

8.5 Further reading

- How to extract information from clusters
- Chapter 10 on community typing

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] ggord_1.1.7                  knitr_1.42
 [3] stringr_1.5.0                patchwork_1.1.2
 [5] scater_1.26.1                scuttle_1.8.4
 [7] ggplot2_3.4.1                vegan_2.6-4
 [9] lattice_0.20-45              permute_0.9-7
```

```
[11] mia_1.7.11                     MultiAssayExperiment_1.24.0
[13] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
[15] XVector_0.38.0                 SingleCellExperiment_1.20.1
[17] SummarizedExperiment_1.28.0    Biobase_2.58.0
[19] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[21] IRanges_2.32.0                 S4Vectors_0.36.2
[23] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[25] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[27] rebook_1.6.0

loaded via a namespace (and not attached):
 [1] Rtsne_0.16                  ggbeeswarm_0.7.1
 [3] colorspace_2.1-0            BiocNeighbors_1.16.0
 [5] farver_2.1.1                ggrepel_0.9.3
 [7] bit64_4.0.5                 fansi_1.0.4
 [9] decontam_1.18.0             codetools_0.2-19
[11] splines_4.2.1               sparseMatrixStats_1.10.0
[13] cachem_1.0.7                jsonlite_1.8.4
[15] cluster_2.1.4               graph_1.74.0
[17] BiocManager_1.30.20         compiler_4.2.1
[19] Matrix_1.5-3                fastmap_1.1.1
[21] lazyeval_0.2.2              cli_3.6.1
[23] BiocSingular_1.14.0         htmltools_0.5.5
[25] tools_4.2.1                 rsvd_1.0.5
[27] gtable_0.3.3                glue_1.6.2
[29] GenomeInfoDbData_1.2.9      reshape2_1.4.4
[31] dplyr_1.1.1                 Rcpp_1.0.10
[33] vctrs_0.6.1                 ape_5.7-1
[35] nlme_3.1-162                DECIPHER_2.26.0
[37] DelayedMatrixStats_1.20.0   xfun_0.38
[39] beachmat_2.14.0             lifecycle_1.0.3
[41] irlba_2.3.5.1               XML_3.99-0.14
[43] zlibbioc_1.44.0             MASS_7.3-58.3
[45] scales_1.2.1                parallel_4.2.1
[47] yaml_2.3.7                  memoise_2.0.1
[49] gridExtra_2.3               yulab.utils_0.0.6
[51] stringi_1.7.12              RSQLite_2.3.0
[53] highr_0.10                  ScaledMatrix_1.6.0
[55] tidytree_0.4.2              filelock_1.0.2
[57] BiocParallel_1.32.6         rlang_1.1.0
[59] pkgconfig_2.0.3             bitops_1.0-7
[61] evaluate_0.20               purrr_1.0.1
[63] treeio_1.22.0               CodeDepends_0.6.5
[65] labeling_0.4.2              cowplot_1.1.1
[67] bit_4.0.5                   tidyselect_1.2.0
[69] plyr_1.8.8                  magrittr_2.0.3
```

```
[71] bookdown_0.33               R6_2.5.1
[73] generics_0.1.3              DelayedArray_0.24.0
[75] DBI_1.1.3                   pillar_1.9.0
[77] withr_2.5.0                 mgcv_1.8-42
[79] RCurl_1.98-1.12             tibble_3.2.1
[81] dir.expiry_1.4.0            crayon_1.5.2
[83] utf8_1.2.3                  rmarkdown_2.21
[85] viridis_0.6.2               grid_4.2.1
[87] blob_1.2.4                  digest_0.6.31
[89] tidyr_1.3.0                 munsell_0.5.0
[91] DirichletMultinomial_1.40.0 beeswarm_0.4.0
[93] viridisLite_0.4.1           vipor_0.4.5
```

Chapter 9

Community composition

```
library(mia)
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
```

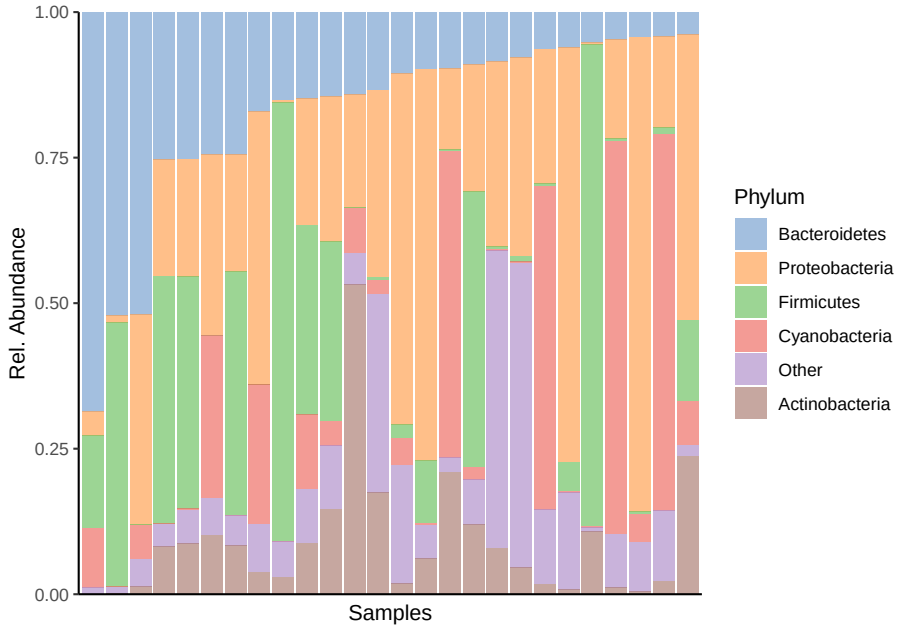
9.1 Visualizing taxonomic composition

9.1.1 Composition barplot

A typical way to visualize microbiome composition is by using composition barplot. In the following, relative abundance is calculated and top taxa are retrieved for the Phylum rank. Thereafter, the barplot is visualized ordering rank by abundance values and samples by "Bacteroidetes":

```
library(miaViz)
# Computing relative abundance
tse <- relAbundanceCounts(tse)

# Getting top taxa on a Phylum level
tse_phylum <- agglomerateByRank(tse, rank ="Phylum",
 ↪  onRankOnly=TRUE)
top_taxa <- getTopTaxa(tse_phylum,top = 5, assay_name =
 ↪  "relabundance")

# Renaming the "Phylum" rank to keep only top taxa and the rest
 ↪  to "Other"
phylum_renamed <- lapply(rowData(tse)$Phylum,
```

```
                    function(x){if (x %in% top_taxa) {x} else
                     ↪  {"Other"}})
rowData(tse)$Phylum <- as.character(phylum_renamed)

# Visualizing the composition barplot, with samples order by
 ↪  "Bacteroidetes"
plotAbundance(tse, assay_name="relabundance", rank = "Phylum",
              order_rank_by="abund",
              order_sample_by = "Bacteroidetes")
```



9.1.2 Composition heatmap

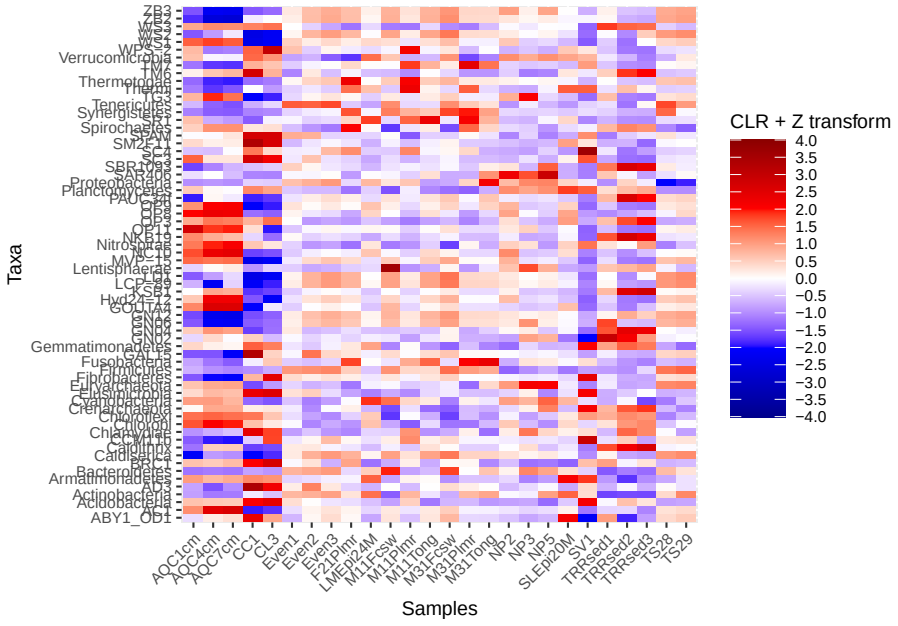
Community composition can be visualized with heatmap, where the horizontal axis represents samples and the vertical axis the taxa. Color of each intersection point represents abundance of a taxon in a specific sample.

Here, abundances are first CLR (centered log-ratio) transformed to remove compositionality bias. Then Z transformation is applied to CLR-transformed data. This shifts all taxa to zero mean and unit variance, allowing visual comparison between taxa that have different absolute abundance levels. After these rough visual exploration techniques, we can visualize the abundances at Phylum level.

```
library(ggplot2)

# Add clr-transformation on samples
assay(tse_phylum, "pseudo") <- assay(tse_phylum, "counts") + 1
tse_phylum <- transformCounts(tse_phylum, assay_name = "pseudo",
                              method = "relabundance")

tse_phylum <- transformCounts(tse_phylum,
                  assay_name = "relabundance",
          method = "clr")

# Add z-transformation on features (taxa)
tse_phylum <- transformCounts(tse_phylum, assay_name = "clr",
                              MARGIN = "features",
                              method = "z", name = "clr_z")
```

Visualize as heatmap.

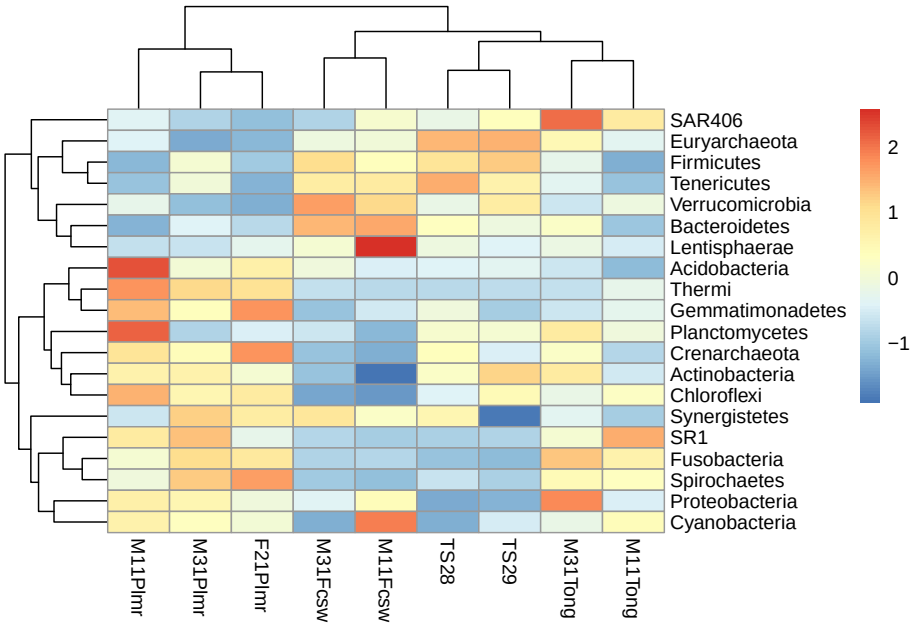
```
# Melt the assay for plotting purposes
df <- meltAssay(tse_phylum, assay_name = "clr_z")

# Determines the scaling of colours
maxval <- round(max(abs(df$clr_z)))
limits <- c(-maxval, maxval)
breaks <- seq(from = min(limits), to = max(limits), by = 0.5)
colours <- c("darkblue", "blue", "white", "red", "darkred")

# Creates a ggplot object
ggplot(df, aes(x = SampleID, y = FeatureID, fill = clr_z)) +
  geom_tile() +
  scale_fill_gradientn(name = "CLR + Z transform",
                       breaks = breaks, limits = limits, colours
                        ↪  = colours) +
  theme(text = element_text(size=10),
        axis.text.x = element_text(angle=45, hjust=1),
        legend.key.size = unit(1, "cm")) +
  labs(x = "Samples", y = "Taxa")
```



pheatmap is a package that provides methods to plot clustered heatmaps.

```
if(!require(pheatmap)){install.packages("pheatmap");
 ↪  library(pheatmap)}

# Takes subset: only samples from feces, skin, or tongue
tse_phylum_subset <- tse_phylum[ , colData(tse_phylum)$SampleType
 ↪  %in% c("Feces", "Skin", "Tongue") ]

# Add clr-transformation
tse_phylum_subset <- transformCounts(tse_phylum_subset,
                          method = "clr",
                  pseudocount = 1)

tse_phylum_subset <- transformCounts(tse_phylum_subset,
 ↪  assay_name = "clr",
                                       MARGIN = "features",
                                       method = "z", name =
                                        ↪  "clr_z")

# Get n most abundant taxa, and subsets the data by them
top_taxa <- getTopTaxa(tse_phylum_subset, top = 20)
tse_phylum_subset <- tse_phylum_subset[top_taxa, ]

# Gets the assay table
```

```
mat <- assay(tse_phylum_subset, "clr_z")

# Creates the heatmap
pheatmap(mat)
```

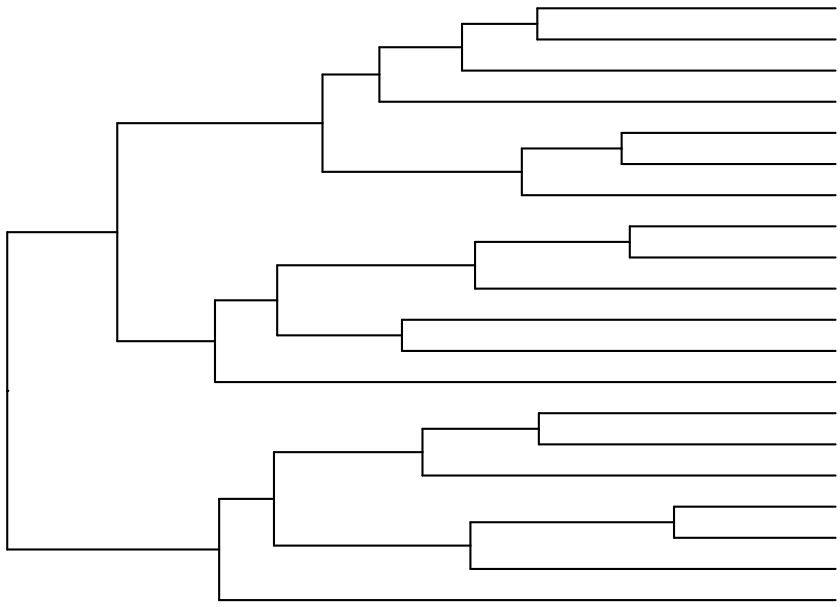


We can create clusters by hierarchical clustering and add them to the plot.

```
if(!require(ape)){
    install.packages("ape")
    library(ape)
}

# Hierarchical clustering
taxa_hclust <- hclust(dist(mat), method = "complete")

# Creates a phylogenetic tree
taxa_tree <- as.phylo(taxa_hclust)
```

```
if(!require(ggtree)){
    install.packages("ggtree")
    library(ggtree)
}
```

```
# Plot taxa tree
taxa_tree <- ggtree(taxa_tree) +
  theme(plot.margin=margin(0,0,0,0)) # removes margins

# Get order of taxa in plot
taxa_ordered <- get_taxa_name(taxa_tree)

taxa_tree
```



Based on phylo tree, we decide to create three clusters.

```
# Creates clusters
taxa_clusters <- cutree(tree = taxa_hclust, k = 3)

# Converts into data frame
taxa_clusters <- data.frame(clusters = taxa_clusters)
taxa_clusters$clusters <- factor(taxa_clusters$clusters)

# Order data so that it's same as in phylo tree
taxa_clusters <- taxa_clusters[taxa_ordered, , drop = FALSE]

# Prints taxa and their clusters
taxa_clusters
```

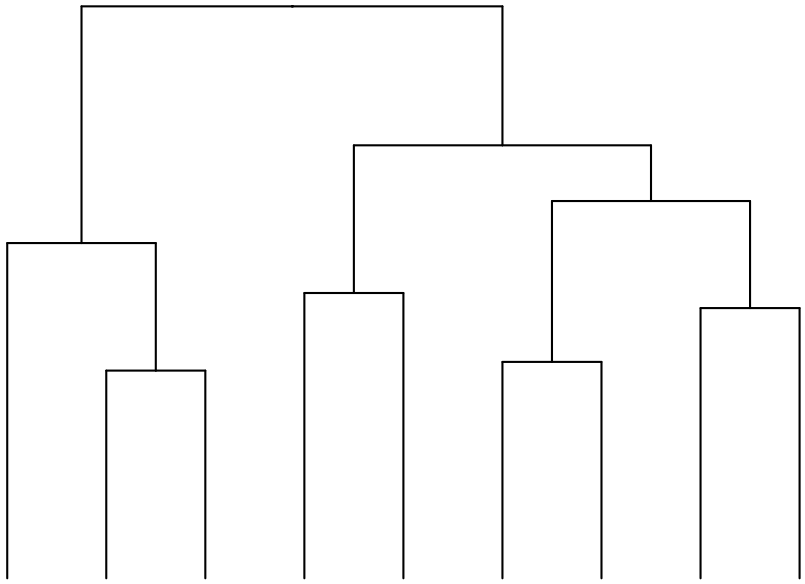
```
##                  clusters
## Chloroflexi             3
## Actinobacteria          3
## Crenarchaeota           3
## Planctomycetes          3
## Gemmatimonadetes        3
## Thermi                  3
## Acidobacteria           3
## Spirochaetes            2
## Fusobacteria            2
## SR1                     2
## Cyanobacteria           2
## Proteobacteria          2
## Synergistetes           2
## Lentisphaerae           1
## Bacteroidetes           1
## Verrucomicrobia         1
## Tenericutes             1
## Firmicutes              1
## Euryarchaeota           1
## SAR406                  1
```

```
# Adds information to rowData
rowData(tse_phylum_subset)$clusters <-
  taxa_clusters[order(match(rownames(taxa_clusters),
  rownames(tse_phylum_subset))), ]

# Prints taxa and their clusters
rowData(tse_phylum_subset)$clusters
```

```
##  [1] 1 1 2 3 2 2 1 1 1 1 3 2 3 3 3 2 2 3 3 1
## Levels: 1 2 3
```

```
# Hierarchical clustering
sample_hclust <- hclust(dist(t(mat)), method = "complete")

# Creates a phylogenetic tree
sample_tree <- as.phylo(sample_hclust)

# Plot sample tree
sample_tree <- ggtree(sample_tree) + layout_dendrogram() +
  theme(plot.margin=margin(0,0,0,0)) # removes margins

# Get order of samples in plot
```

```
samples_ordered <- rev(get_taxa_name(sample_tree))

sample_tree
```



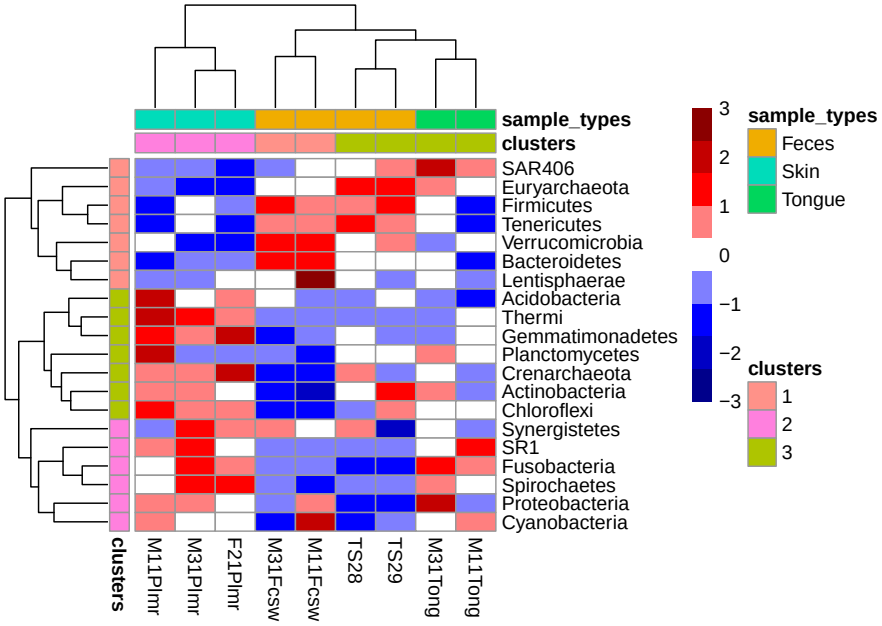
```
# Creates clusters
sample_clusters <- factor(cutree(tree = sample_hclust, k = 3))

# Converts into data frame
sample_data <- data.frame(clusters = sample_clusters)

# Order data so that it's same as in phylo tree
sample_data <- sample_data[samples_ordered, , drop = FALSE]

# Order data based on
tse_phylum_subset <- tse_phylum_subset[ , rownames(sample_data)]

# Add sample type data
sample_data$sample_types <-
 ↪  unfactor(colData(tse_phylum_subset)$SampleType)

sample_data
```

```
##          clusters sample_types
```

```
## M11Plmr        2         Skin
## M31Plmr        2         Skin
## F21Plmr        2         Skin
## M31Fcsw        1        Feces
## M11Fcsw        1        Feces
## TS28           3        Feces
## TS29           3        Feces
## M31Tong        3       Tongue
## M11Tong        3       Tongue
```

Now we can create heatmap with additional annotations.

```
# Determines the scaling of colorss
# Scale colors
breaks <- seq(-ceiling(max(abs(mat))), ceiling(max(abs(mat))),
              length.out = ifelse( max(abs(mat))>5,
               ↪  2*ceiling(max(abs(mat))), 10 ) )
colors <- colorRampPalette(c("darkblue", "blue", "white", "red",
 ↪  "darkred"))(length(breaks)-1)

pheatmap(mat, annotation_row = taxa_clusters,
         annotation_col = sample_data,
         breaks = breaks,
         color = colors)
```



In addition, there are also other packages that provide functions for more complex heatmaps, such as *iheatmapr* and ComplexHeatmap (Gu, 2022). sechm package provides wrapper for *ComplexHeatmap* and its usage is explained in chapter 14 along with the `pheatmap` package for clustered heatmaps.

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] ggtree_3.4.4                    ape_5.7-1
 [3] pheatmap_1.0.12                 miaViz_1.7.5
 [5] ggraph_2.1.0                    ggplot2_3.4.1
 [7] mia_1.7.11                      MultiAssayExperiment_1.24.0
 [9] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
[11] XVector_0.38.0                  SingleCellExperiment_1.20.1
[13] SummarizedExperiment_1.28.0     Biobase_2.58.0
[15] GenomicRanges_1.50.2            GenomeInfoDb_1.34.9
[17] IRanges_2.32.0                  S4Vectors_0.36.2
[19] BiocGenerics_0.44.0             MatrixGenerics_1.10.0
[21] matrixStats_0.63.0-9003         BiocStyle_2.24.0
[23] rebook_1.6.0

loaded via a namespace (and not attached):
```

```
 [1] ggnewscale_0.4.8            ggbeeswarm_0.7.1
 [3] colorspace_2.1-0            scuttle_1.8.4
 [5] BiocNeighbors_1.16.0        aplot_0.1.10
 [7] farver_2.1.1                graphlayouts_0.8.4
 [9] ggrepel_0.9.3               bit64_4.0.5
[11] fansi_1.0.4                 decontam_1.18.0
[13] codetools_0.2-19            splines_4.2.1
[15] sparseMatrixStats_1.10.0    cachem_1.0.7
[17] knitr_1.42                  scater_1.26.1
[19] polyclip_1.10-4             jsonlite_1.8.4
[21] cluster_2.1.4               graph_1.74.0
[23] ggforce_0.4.1               BiocManager_1.30.20
[25] compiler_4.2.1              Matrix_1.5-3
[27] fastmap_1.1.1               lazyeval_0.2.2
[29] cli_3.6.1                   tweenr_2.0.2
[31] BiocSingular_1.14.0         htmltools_0.5.5
[33] tools_4.2.1                 igraph_1.4.1
[35] rsvd_1.0.5                  gtable_0.3.3
[37] glue_1.6.2                  GenomeInfoDbData_1.2.9
[39] reshape2_1.4.4              dplyr_1.1.1
[41] Rcpp_1.0.10                 vctrs_0.6.1
[43] nlme_3.1-162                DECIPHER_2.26.0
[45] DelayedMatrixStats_1.20.0   xfun_0.38
[47] stringr_1.5.0               beachmat_2.14.0
[49] lifecycle_1.0.3             irlba_2.3.5.1
[51] XML_3.99-0.14               zlibbioc_1.44.0
[53] MASS_7.3-58.3               scales_1.2.1
[55] tidygraph_1.2.3             parallel_4.2.1
[57] RColorBrewer_1.1-3          yaml_2.3.7
[59] memoise_2.0.1               gridExtra_2.3
[61] ggfun_0.0.9                 yulab.utils_0.0.6
[63] stringi_1.7.12              RSQLite_2.3.0
[65] highr_0.10                  ScaledMatrix_1.6.0
[67] tidytree_0.4.2              permute_0.9-7
[69] filelock_1.0.2              BiocParallel_1.32.6
[71] rlang_1.1.0                 pkgconfig_2.0.3
[73] bitops_1.0-7                evaluate_0.20
[75] lattice_0.20-45             purrr_1.0.1
[77] labeling_0.4.2              patchwork_1.1.2
[79] treeio_1.22.0               CodeDepends_0.6.5
[81] bit_4.0.5                   tidyselect_1.2.0
[83] plyr_1.8.8                  magrittr_2.0.3
[85] bookdown_0.33               R6_2.5.1
[87] generics_0.1.3              DelayedArray_0.24.0
[89] DBI_1.1.3                   pillar_1.9.0
[91] withr_2.5.0                 mgcv_1.8-42
```

```
 [93] RCurl_1.98-1.12             tibble_3.2.1
 [95] dir.expiry_1.4.0            crayon_1.5.2
 [97] utf8_1.2.3                  rmarkdown_2.21
 [99] viridis_0.6.2               grid_4.2.1
[101] blob_1.2.4                  vegan_2.6-4
[103] digest_0.6.31               tidyr_1.3.0
[105] gridGraphics_0.5-1          munsell_0.5.0
[107] DirichletMultinomial_1.40.0 ggplotify_0.1.0
[109] beeswarm_0.4.0              viridisLite_0.4.1
[111] vipor_0.4.5
```

Chapter 10

Community typing (clustering)

```
library(mia)
data("GlobalPatterns", package="mia")
tse <- GlobalPatterns
```

Clustering is an unsupervised machine learning technique. The idea of it is to find clusters from the data. A cluster is a group of features/samples that share pattern. For example, with clustering, we can find group of samples that share similar community composition. There are multiple clustering algorithms available.

10.1 Hiearchical clustering

Hiearchical clustering is a clustering method that aims to find hiearchy between samples/features. There are to approaches: agglomerative ("bottom-up") and divisive ("top-down").

In agglomerative approach, each observation is first unique cluster. Algorithm continues by agglomerating similar clusters. Divisive approach starts with one cluster that contains all the observations. Clusters are splitted recursively to clusters that differ the most. Clustering ends when each cluster contains only one observation.

Hiearchical clustering can be visualized with dendrogram tree. In each splitting point, the three is divided into two clusters leading to hierarchy.

Let's load data from mia package.

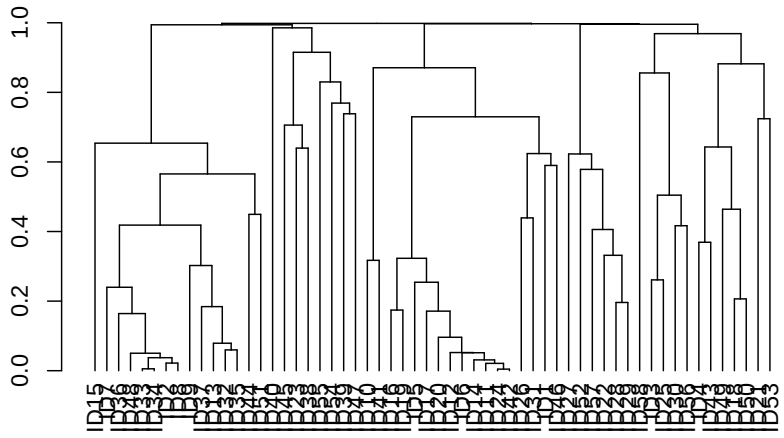
```
library(mia)
library(vegan)

# Load experimental data
data(peerj13075)
(tse <- peerj13075)
```

```
## class: TreeSummarizedExperiment
## dim: 674 58
## metadata(0):
## assays(1): counts
## rownames(674): OTU1 OTU2 ... OTU2567 OTU2569
## rowData names(6): kingdom phylum ... family genus
## colnames(58): ID1 ID2 ... ID57 ID58
## colData names(5): Sample Geographical_location Gender Age Diet
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

Hierarchical clustering requires 2 steps. In the fist step, dissimilarities are calculated. In prior to that, data transformation is applied if needed. Since sequencing data is compositional, relative transformation is applied. In the second step, clustering is performed based on dissimilarities.

```
if( !require(NbClust) ){install.packages("NbClust");
 ↪  library(NbClust)}
if( !require(cobiclust) ){install.packages("cobiclust");
 ↪  library(cobiclust)}

# Apply transformation
tse <- transformCounts(tse, method = "relabundance")
# Get the assay
assay <- assay(tse, "relabundance")
# Transpose assay --> samples are now in rows --> we are
 ↪  clustering samples
assay <- t(assay)

# Calculate distances
diss <- vegdist(assay, method = "bray")
```

```
# Perform hierarchical clustering
hc <- hclust(diss, method = "complete")

# To visualize, convert hclust object into dendrogram object
dendro <- as.dendrogram(hc)

# Plot dendrogram
plot(dendro)
```



We can use dendrogram to determine the number of clusters. Usually the tree is splitted where the branch length is the largest. However, as we can see from the dendrogram, clusters are not clear. Algorithms are available to identify the optimal number of clusters.

```
# Determine the optimal number of clusters
res <- NbClust(diss = diss, distance = NULL, method = "ward.D2",
               index = "silhouette")
```

```
##
##  Only frey, mcclain, cindex, sihouette and dunn can be computed. To compute the other indices,
```

```
res$Best.nc
```

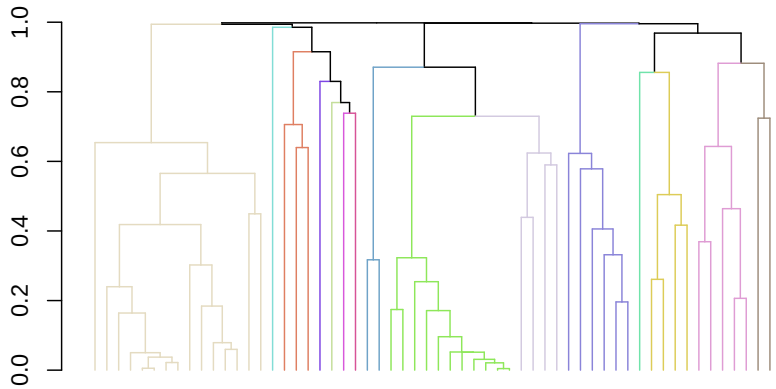
```
## Number_clusters     Value_Index
##         15.0000          0.4543
```

Based on the result, let's divide observations into 15 clusters.

```
if( !require(dendextend) ){
    install.packages("dendextend")
    library(dendextend)
}

# Find clusters
cutree(hc, k = 15)
```

```
##  ID1  ID2  ID3  ID4  ID5  ID6  ID7  ID8  ID9 ID10 ID11 ID12 ID13 ID14 ID15 ID16
##    1    2    3    4    5    5    2    2    2    6    5    5    2    5    2    5
## ID17 ID18 ID19 ID20 ID21 ID22 ID23 ID24 ID25 ID26 ID27 ID28 ID29 ID30 ID31 ID32
##    5    4    5    5    7    8    9    5    3    1    8    8    8    3    1    2
## ID33 ID34 ID35 ID36 ID37 ID38 ID39 ID40 ID41 ID42 ID43 ID44 ID45 ID46 ID47 ID48
##    2    2    2    2    2    9   10   11    6    5    4    2    9    1   12    2
## ID49 ID50 ID51 ID52 ID53 ID54 ID55 ID56 ID57 ID58
##    4    4    2    8    7   13   14    3    8   15
```

```
# Making colors for 6 clusters
col_val_map <- randomcoloR::distinctColorPalette(15) %>%
     as.list() %>% setNames(paste0("clust_",seq(15)))

dend <- color_branches(dendro, k=15, col=unlist(col_val_map))
labels(dend) <- NULL
plot(dend)
```



10.2 K-means clustering

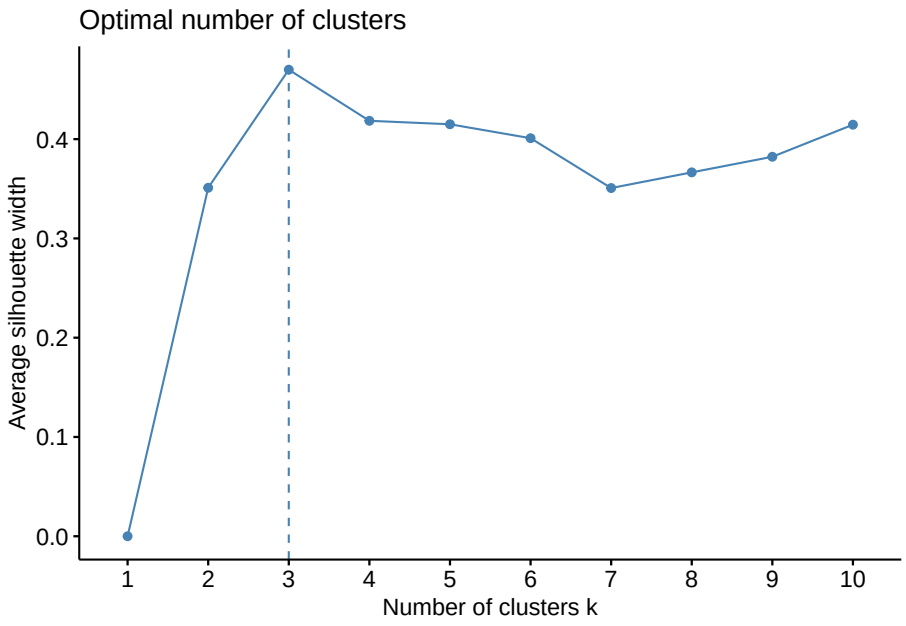
Hierarchical clustering did not yield clusters. Let's try k-means clustering instead. Here observations are divided into clusters so that the distances between observations and cluster centers are minimized; an observation belongs to cluster whose center is the nearest.

The algorithm starts by dividing observation to random clusters whose number is defined by user. The centroids of clusters are then calculated. After that, observations' allocation to clusters are updated so that the means are minimized. Again, centroid are calculated, and algorithm continues iteratively until the assignments do not change.

The number of clusters can be determined based on algorithm. Here we utilize silhouette analysis.

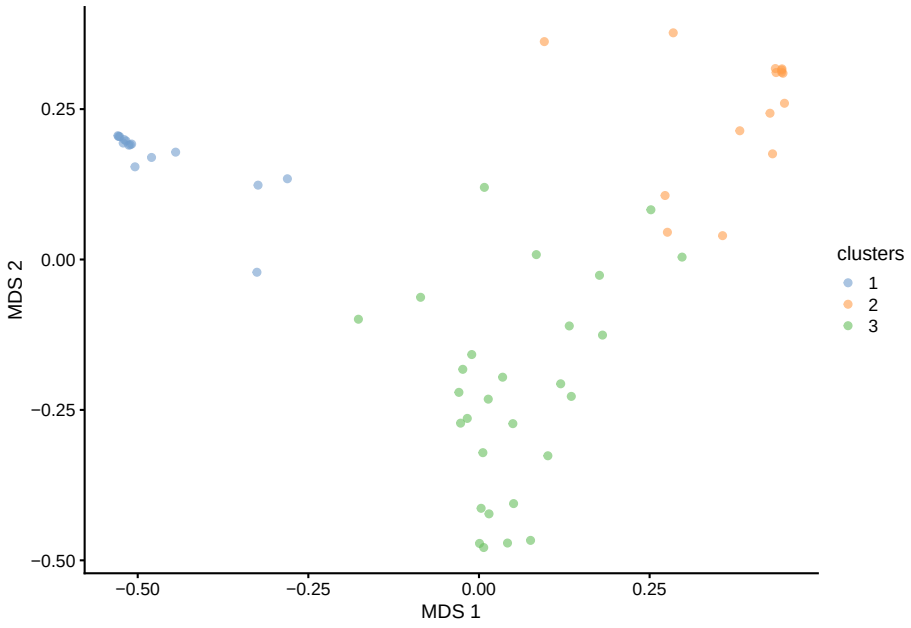
```
if( !require(factoextra) ){
    install.packages("factoextra")
    library(factoextra)
}

# Convert dist object into matrix
diss <- as.matrix(diss)
```

```
# Perform silhouette analysis and plot the result
fviz_nbclust(diss, kmeans, method = "silhouette")
```



Based on the result of silhouette analysis, we choose 3 to be the number of clusters in k-means clustering.

```
library(scater)

# The first step is random, add seed for reproducibility
set.seed(15463)
# Perform k-means clustering with 3 clusters
km <- kmeans(diss, 3, nstart = 25)
# Add the result to colData
colData(tse)$clusters <- as.factor(km$cluster)

# Perform PCoA so that we can visualize clusters
tse <- runMDS(tse, exprs_values = "relabundance", FUN =
 ↪  vegan::vegdist, method = "bray")

# Plot PCoA and color clusters
plotReducedDim(tse, "MDS", colour_by = "clusters")
```



10.3 Dirichlet Multinomial Mixtures (DMM)

This section focus on DMM analysis.

One technique that allows to search for groups of samples that are similar to each other is the Dirichlet-Multinomial Mixture Model. In DMM, we first determine the number of clusters (k) that best fit the data (model evidence) using Laplace approximation. After fitting the model with k clusters, we obtain for each sample k probabilities that reflect the probability that a sample belongs to the given cluster.

Let's cluster the data with DMM clustering.

```
# Runs model and calculates the most likely number of clusters
 ↪  from 1 to 7.
# Since this is a large dataset it takes long computational time.
# For this reason we use only a subset of the data; agglomerated
 ↪  by Phylum as a rank.
tse <- GlobalPatterns
tse <- agglomerateByRank(tse, rank = "Phylum",
 ↪  agglomerateTree=TRUE)
```

```
tse_dmn <- mia::runDMN(tse, name = "DMN", k = 1:7)
```

```
# It is stored in metadata
tse_dmn
```

```
## class: TreeSummarizedExperiment
## dim: 67 26
## metadata(2): agglomerated_by_rank DMN
## assays(1): counts
## rownames(67): Phylum:Crenarchaeota Phylum:Euryarchaeota ...
##   Phylum:Synergistetes Phylum:SR1
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(26): CL3 CC1 ... Even2 Even3
## colData names(7): X.SampleID Primer ... SampleType Description
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (67 rows)
## rowTree: 1 phylo tree(s) (66 leaves)
## colLinks: NULL
## colTree: NULL
```

Return information on metadata that the object contains.

```
names(metadata(tse_dmn))
```

```
## [1] "agglomerated_by_rank" "DMN"
```

This returns a list of DMN objects for a closer investigation.

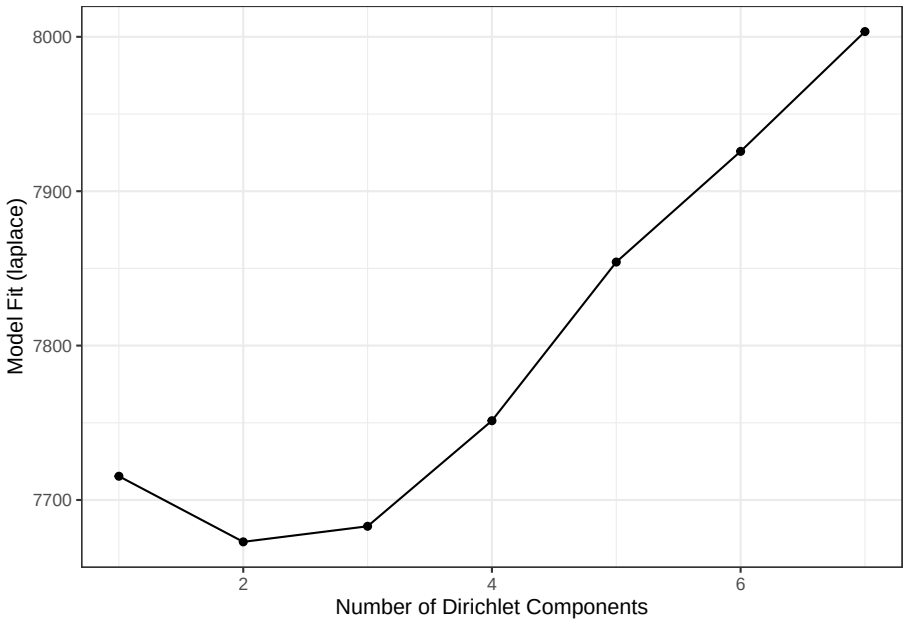
```
getDMN(tse_dmn)
```

```
## [[1]]
## class: DMN
## k: 1
## samples x taxa: 26 x 67
## Laplace: 7715 BIC: 7802 AIC: 7760
##
## [[2]]
## class: DMN
## k: 2
```

```
## samples x taxa: 26 x 67
## Laplace: 7673 BIC: 7927 AIC: 7842
##
## [[3]]
## class: DMN
## k: 3
## samples x taxa: 26 x 67
## Laplace: 7683 BIC: 8069 AIC: 7942
##
## [[4]]
## class: DMN
## k: 4
## samples x taxa: 26 x 67
## Laplace: 7751 BIC: 8274 AIC: 8103
##
## [[5]]
## class: DMN
## k: 5
## samples x taxa: 26 x 67
## Laplace: 7854 BIC: 8553 AIC: 8340
##
## [[6]]
## class: DMN
## k: 6
## samples x taxa: 26 x 67
## Laplace: 7926 BIC: 8796 AIC: 8540
##
## [[7]]
## class: DMN
## k: 7
## samples x taxa: 26 x 67
## Laplace: 8003 BIC: 9051 AIC: 8752
```

Show Laplace approximation (model evidence) for each model of the k models.

```
library(miaViz)
plotDMNFit(tse_dmn, type = "laplace")
```



Return the model that has the best fit.

```
getBestDMNFit(tse_dmn, type = "laplace")
```

```
## class: DMN
## k: 2
## samples x taxa: 26 x 67
## Laplace: 7673 BIC: 7927 AIC: 7842
```

10.3.1 PCoA for ASV-level data with Bray-Curtis; with DMM clusters shown with colors

Group samples and return DMNGroup object that contains a summary. Patient status is used for grouping.

```
dmn_group <- calculateDMNgroup(tse_dmn, variable = "SampleType",
 ↪  exprs_values = "counts",
                               k = 2, seed=.Machine$integer.max)

dmn_group
```

```
## class: DMNGroup
```

```
## summary:
##                     k samples taxa    NLE  LogDet Laplace    BIC  AIC
## Feces               2       4   67 1078.3 -106.26   901.1 1171.9 1213
## Freshwater          2       2   67  889.6  -97.23   716.9  936.4 1025
## Freshwater (creek) 2       3   67 1600.3  793.17  1872.8 1674.5 1735
## Mock                2       3   67  998.6  -70.65   839.2 1072.8 1134
## Ocean               2       3   67 1096.7  -56.66   944.3 1170.9 1232
## Sediment (estuary) 2       3   67 1195.5   18.63  1080.8 1269.7 1331
## Skin                2       3   67  992.6  -85.05   826.1 1066.8 1128
## Soil                2       3   67 1380.3   11.20  1261.8 1454.5 1515
## Tongue              2       2   67  783.0 -107.79   605.0  829.8  918
```

Mixture weights (rough measure of the cluster size).

```
DirichletMultinomial::mixturewt(getBestDMNFit(tse_dmn))
```

```
##       pi theta
## 1 0.5385 20.58
## 2 0.4615 15.28
```

Samples-cluster assignment probabilities / how probable it is that sample belongs to each cluster

```
head(DirichletMultinomial::mixture(getBestDMNFit(tse_dmn)))
```

```
##              [,1]      [,2]
## CL3     1.000e+00 5.050e-17
## CC1     1.000e+00 3.903e-22
## SV1     1.000e+00 1.957e-12
## M31Fcsw 7.886e-26 1.000e+00
## M11Fcsw 1.132e-16 1.000e+00
## M31Plmr 1.124e-13 1.000e+00
```

Contribution of each taxa to each component

```
head(DirichletMultinomial::fitted(getBestDMNFit(tse_dmn)))
```

```
##                           [,1]      [,2]
## Phylum:Crenarchaeota  0.30382 0.1354654
## Phylum:Euryarchaeota  0.23114 0.1468632
## Phylum:Actinobacteria 1.21371 1.0600245
## Phylum:Spirochaetes   0.21393 0.1318415
## Phylum:MVP-15         0.02982 0.0007669
## Phylum:Proteobacteria 6.84469 1.8153216
```

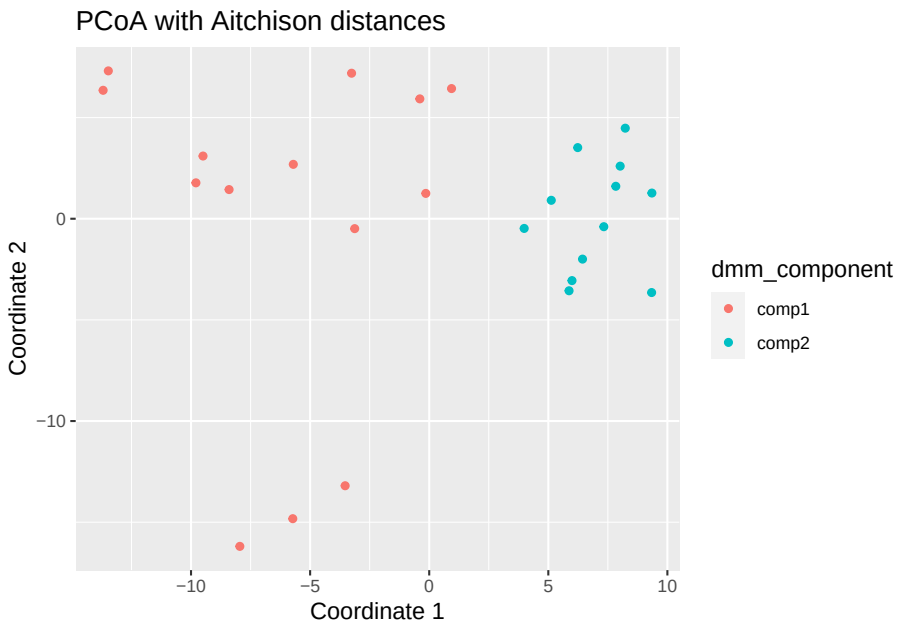
Get the assignment probabilities

```
prob <- DirichletMultinomial::mixture(getBestDMNFit(tse_dmn))
# Add column names
colnames(prob) <- c("comp1", "comp2")

# For each row, finds column that has the highest value. Then
 ↪  extract the column
# names of highest values.
vec <- colnames(prob)[max.col(prob,ties.method = "first")]
```

Computing the euclidean PCoA and storing it as a data frame

```
# Does clr transformation. Pseudocount is added, because data
 ↪  contains zeros.
assay(tse, "pseudo") <- assay(tse, "counts") + 1
tse <- transformCounts(tse, assay_name = "pseudo", method =
 ↪  "relabundance")
tse <- transformCounts(tse, "relabundance", method = "clr")

library(scater)

# Does principal coordinate analysis
df <- calculateMDS(tse, exprs_values = "clr", method =
 ↪  "euclidean")

# Creates a data frame from principal coordinates
euclidean_pcoa_df <- data.frame(pcoa1 = df[,1],
                                pcoa2 = df[,2])
```

```
# Creates a data frame that contains principal coordinates and
 ↪  DMM information
euclidean_dmm_pcoa_df <- cbind(euclidean_pcoa_df,
                               dmm_component = vec)
# Creates a plot
euclidean_dmm_plot <- ggplot(data = euclidean_dmm_pcoa_df,
                             aes(x=pcoa1, y=pcoa2,
                                 color = dmm_component)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "PCoA with Aitchison distances") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_dmm_plot
```



10.4 Community Detection

Another approach for discovering communities within the samples of the data, is to run community detection algorithms after building a graph. The following demonstration builds a graph based on the k nearest-neighbors and performs the community detection on the fly.

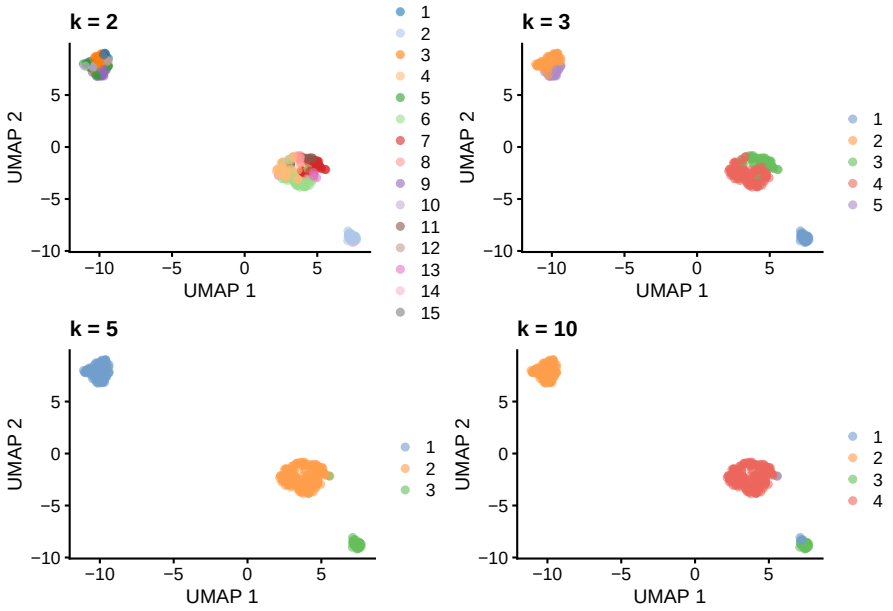
bluster (Lun, 2021) package offers several clustering methods, among which graph-based are present, enabling the community detection task.

Installing package:

```
if(!require(bluster)){
  BiocManager::install("bluster")
}
```

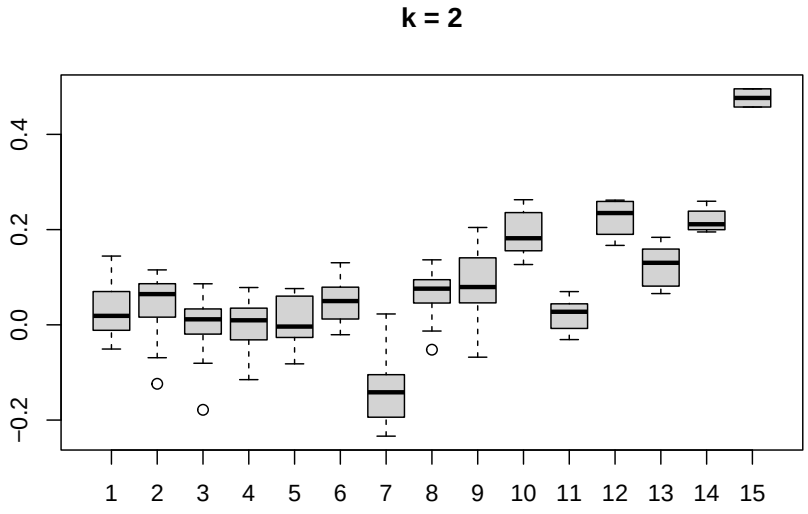
The algorithm used is "short random walks" (Pons and Latapy, 2006). Graph is constructed using different k values (the number of nearest neighbors to consider during graph construction) using the robust centered log ratio (rclr) assay data. Then plotting the communities using UMAP (McInnes et al., 2018) ordination as a visual exploration aid. In the following demonstration we use the `enterotype` dataset from the (Ernst et al., 2020) package.

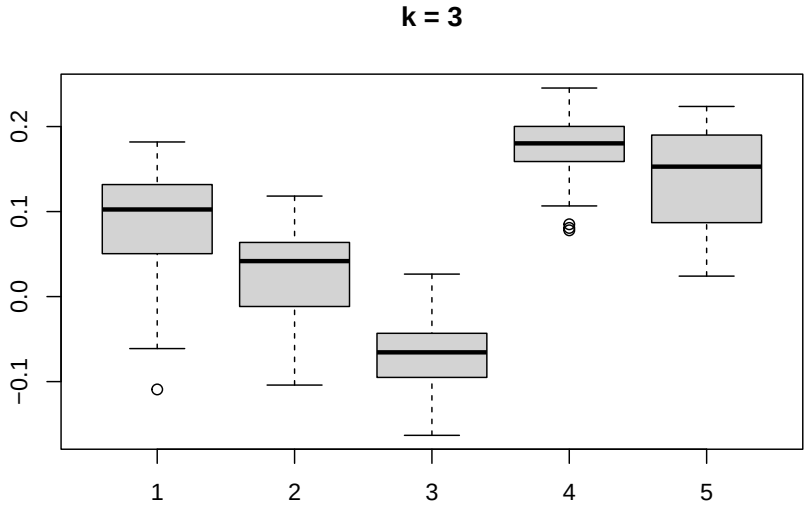
```
library(bluster)
library(patchwork) # For arranging several plots as a grid
library(scater)

data("enterotype", package="mia")
tse <- enterotype
tse <- transformCounts(tse, method = "rclr")

# Performing and storing UMAP
tse <- runUMAP(tse, name="UMAP", exprs_values="rclr")

k <- c(2,3,5,10)
ClustAndPlot <- function(x) {
  # Creating the graph and running the short random walks
   ↪  algorithm
  graph_clusters <- clusterRows(t(assays(tse)$rclr),
 ↪  NNGraphParam(k=x))

  # Results of the clustering as a color for each sample
  plotUMAP(tse, colour_by = I(graph_clusters)) +
    labs(title = paste0("k = ", x))
}

# Applying the function for different k values
plots <- lapply(k,ClustAndPlot)

# Displaying plots in a grid
(plots[[1]] + plots[[2]]) / (plots[[3]] + plots[[4]])
```

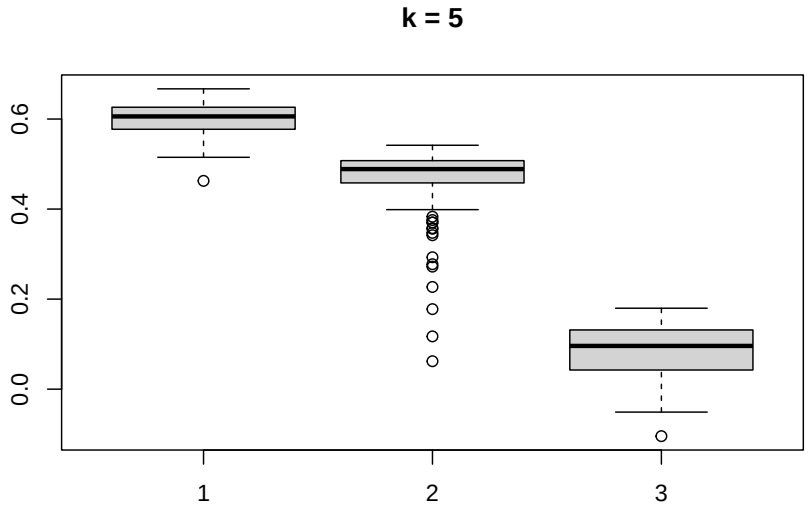


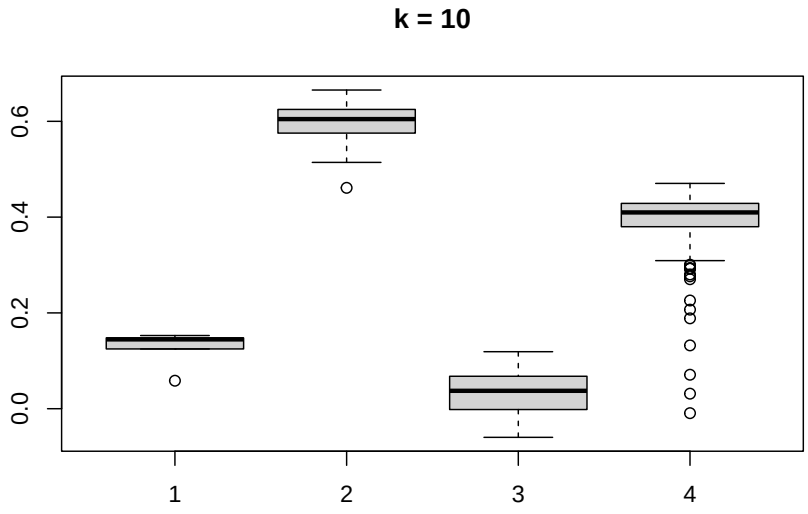
Similarly, the *bluster* (Lun, 2021) package offers clustering diagnostics that can be used for judging the clustering quality (see Assorted clustering diagnostics). In the following, Silhouette width as a diagnostic tool is computed and results are visualized for each case presented earlier. For more about Silhouettes read (Rousseeuw, 1987).

```
ClustDiagPlot <- function(x) {
  # Getting the clustering results
  graph_clusters <- clusterRows(t(assays(tse)$rclr),
 ↪  NNGraphParam(k=x))

  # Computing the diagnostic info
  sil <- approxSilhouette(t(assays(tse)$rclr), graph_clusters)

  # Plotting as a boxlpot to observe cluster separation
  boxplot(split(sil$width, graph_clusters), main=paste0("k = ",
   ↪  x))

}
# Applying the function for different k values
res <- lapply(k,ClustDiagPlot)
```


k = 2
0.4
0.2
0.0
−0.2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


k = 3
0.2
0.1
0.0
−0.1
1 2 3 4 5


k = 5
0.0
0.2
0.4
0.6
1
2
3


k = 10
0.0
0.2
0.4
0.6
1
2
3
4

10.5 Biclustering

Biclustering methods cluster rows and columns simultaneously in order to find subsets of correlated features/samples.

Here, we use following packages:

- biclust
- cobiclust

cobiclust is especially developed for microbiome data whereas *biclust* is more general method. In this section, we show three different cases and example solutions to apply biclustering to them.

1. Taxa vs samples
2. Taxa vs biomolecule/biomarker
3. Taxa vs taxa

Biclusters can be visualized using heatmap or boxplot, for instance. For checking purposes, also scatter plot might be valid choice.

Check more ideas for heatmaps from chapters 14 and @ref(microbiome-community.

10.5.1 Taxa vs samples

When you have microbial abundance matrices, we suggest to use *cobiclust* which is designed for microbial data.

Load example data

```
library(mia)
data("HintikkaXOData")
mae <- HintikkaXOData
```

Only the most prevalent taxa are included in analysis.

```
# Subset data in the first experiment
mae[[1]] <- subsetByPrevalentTaxa(mae[[1]], rank = "Genus",
 ↪  prevalence = 0.2, detection = 0.001)
# clr-transform in the first experiment
mae[[1]] <- transformCounts(mae[[1]], method = "relabundance")
mae[[1]] <- transformCounts(mae[[1]], "relabundance", method =
 ↪  "rclr")
```

cobiclust takes counts table as an input and gives *cobiclust* object as an output. It includes clusters for taxa and samples.

```
# Do clustering; use counts table´
clusters <- cobiclust(assay(mae[[1]], "counts"))

# Get clusters
row_clusters <- clusters$classification$rowclass
col_clusters <- clusters$classification$colclass

# Add clusters to rowdata and coldata
rowData(mae[[1]])$clusters <- factor(row_clusters)
colData(mae[[1]])$clusters <- factor(col_clusters)

# Order data based on clusters
mae[[1]] <- mae[[1]][order(rowData(mae[[1]])$clusters),
 ↪  order(colData(mae[[1]])$clusters)]

# Print clusters
clusters$classification
```

```
## $rowclass
##  [1] 1 1 1 1 2 2 1 1 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 2 1 1 2 1 1 1 1 1 2 1 1 2 1 1
## [39] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1
##
## $colclass
##  C1  C2  C3  C4  C5  C6  C7  C8  C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20
##   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31 C32 C33 C34 C35 C36 C37 C38 C39 C40
##   2   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   1
```

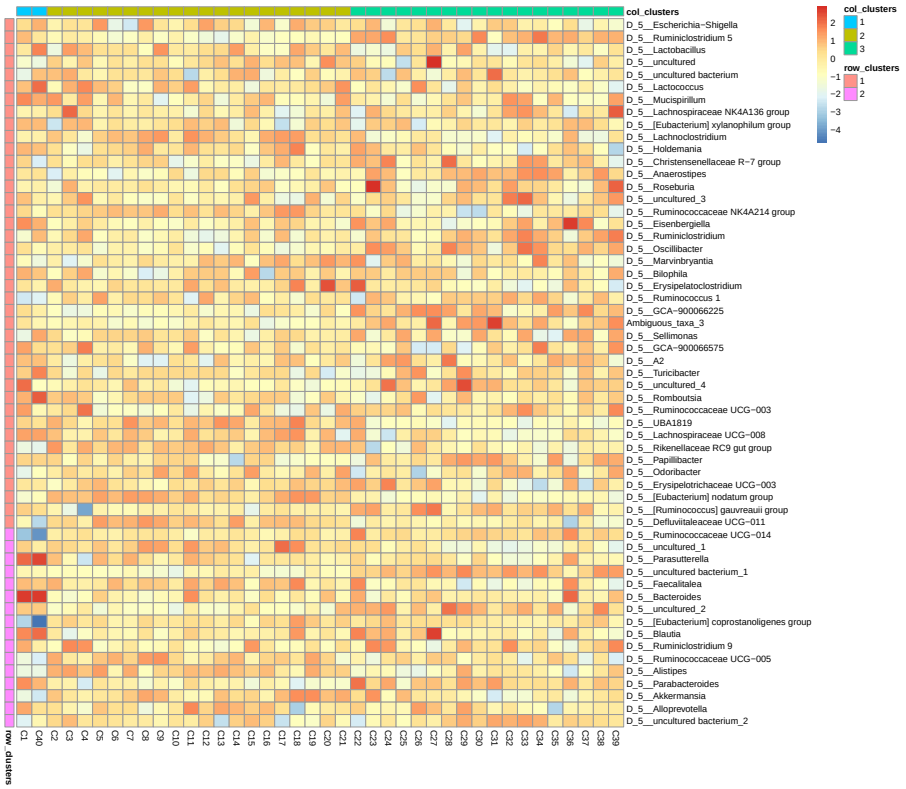
Next we can plot clusters. Annotated heatmap is a common choice.

```
if(!require(pheatmap)){
    install.packages("pheatmap")
    library(pheatmap)
}
# z-transform for heatmap
mae[[1]] <- transformCounts(mae[[1]], assay_name = "rclr",
                            MARGIN = "features",
                            method = "z", name = "clr_z")

# Create annotations. When column names are equal, they should
 ↪  share levels.
# Here samples include 3 clusters, and taxa 2. That is why we
 ↪  have to make
```

```
# column names unique.
annotation_col <- data.frame(colData(mae[[1]])[, "clusters", drop
 ↪  = F])
colnames(annotation_col) <- "col_clusters"

annotation_row <- data.frame(rowData(mae[[1]])[, "clusters", drop
 ↪  = F])
colnames(annotation_row) <- "row_clusters"
```

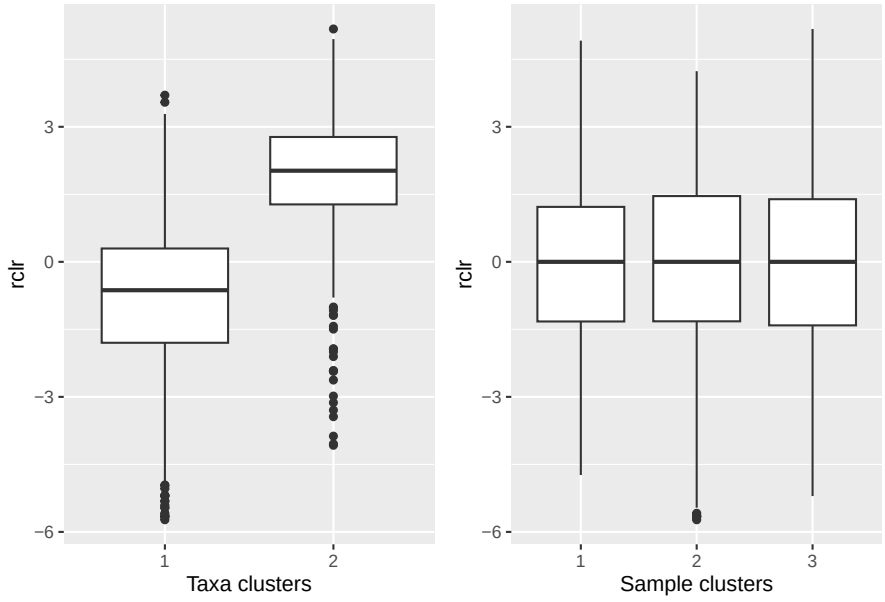
Plot the heatmap.

```
pheatmap(assay(mae[[1]], "clr_z"), cluster_rows = F, cluster_cols
 ↪  = F,
         annotation_col = annotation_col,
         annotation_row = annotation_row)
```



Boxplot is commonly used to summarize the results:

```
if(!require(ggplot2)){
    install.packages("ggplot2")
    library(ggplot2)
}
if(!require(patchwork)){
    install.packages("patchwork")
    library(patchwork)
}

# ggplot requires data in melted format
melt_assay <- meltAssay(mae[[1]], assay_name = "rclr",
 ↪  add_col_data = T, add_row_data = T)

# patchwork two plots side-by-side
p1 <- ggplot(melt_assay) +
  geom_boxplot(aes(x = clusters.x, y = rclr)) +
  labs(x = "Taxa clusters")

p2 <- ggplot(melt_assay) +
  geom_boxplot(aes(x = clusters.y, y = rclr)) +
  labs(x = "Sample clusters")

p1 + p2
```



10.5.2 Taxa vs biomolecules

Here, we analyze cross-correlation between taxa and metabolites. This is a case, where we use *biclust* method which is suitable for numeric matrices in general.

```
# Samples must be in equal order
# (Only 1st experiment  was ordered in cobiclust step leading to
 ↪  unequal order)
mae[[1]] <- mae[[1]][ , colnames(mae[[2]]) ]

# Make rownames unique since it is require by other steps
rownames(mae[[1]]) <- make.unique(rownames(mae[[1]]))
# Calculate correlations
corr <- getExperimentCrossCorrelation(mae, 1, 2,
                                      assay_name1 = "rclr",
                                      assay_name2 = "nmr",
                                      mode = "matrix",
                                      cor_threshold = 0.2)
```

biclust takes matrix as an input and returns *biclust* object.

```
# Load package
if(!require(biclust)){
    install.packages("biclust")
    library(biclust)
}

# Set seed for reproducibility
set.seed(3973)

# Find biclusters
bc <- biclust(corr, method=BCPlaid(), fit.model = y ~ m,
              background = TRUE, shuffle = 100, back.fit = 0,
               ↪  max.layers = 10,
              iter.startup = 10, iter.layer = 100, verbose =
               ↪  FALSE)

bc
```

```
##
## An object of class Biclust
##
## call:
##  biclust(x = corr, method = BCPlaid(), fit.model = y ~ m, background = TRUE,
```

```
##      shuffle = 100, back.fit = 0, max.layers = 10, iter.startup = 10,
##      iter.layer = 100, verbose = FALSE)
##
## There was no cluster found
```

The object includes cluster information. However compared to *cobiclust*, *biclust* object includes only information about clusters that were found, not general cluster.

Meaning that if one cluster size of 5 features was found out of 20 features, those 15 features do not belong to any cluster. That is why we have to create an additional cluster for features/samples that are not assigned into any cluster.

```
# Functions for obtaining biclust information

# Get clusters for rows and columns
.get_biclusters_from_biclust <- function(bc, assay){
  # Get cluster information for columns and rows
  bc_columns <- t(bc@NumberxCol)
  bc_columns <- data.frame(bc_columns)
  bc_rows <- bc@RowxNumber
  bc_rows <- data.frame(bc_rows)

  # Get data into right format
  bc_columns <- .manipulate_bc_data(bc_columns, assay, "col")
  bc_rows <- .manipulate_bc_data(bc_rows, assay, "row")

  return(list(bc_columns = bc_columns, bc_rows = bc_rows))
}

# Input clusters, and how many observations there should be,
 ↪  i.e.,
# the number of samples or features
.manipulate_bc_data <- function(bc_clusters, assay, row_col){
  # Get right dimension
  dim <- ifelse(row_col == "col", ncol(assay), nrow(assay))
  # Get column/row names
  if( row_col == "col" ){
    names <- colnames(assay)
  } else{
    names <- rownames(assay)
  }

  # If no clusters were found, create one. Otherwise create
   ↪  additional
  # cluster which
```

```
  # contain those samples that are not included in clusters that
   ↪ were found.
  if( nrow(bc_clusters) != dim ){
      bc_clusters <- data.frame(cluster = rep(TRUE, dim))
  } else {
      # Create additional cluster that includes those
       ↪ samples/features that
      # are not included in other clusters.
      vec <- ifelse(rowSums(bc_clusters) > 0, FALSE, TRUE)
      # If additional cluster contains samples, then add it
      if ( any(vec) ){
          bc_clusters <- cbind(bc_clusters, vec)
      }
  }
  # Adjust row and column names
  rownames(bc_clusters) <- names
  colnames(bc_clusters) <- paste0("cluster_",
   ↪ 1:ncol(bc_clusters))
  return(bc_clusters)
}
```

```
# Get biclusters
bcs <- .get_biclusters_from_biclust(bc, corr)

bicluster_rows <- bcs$bc_rows
bicluster_columns <- bcs$bc_columns

# Print biclusters for rows
head(bicluster_rows)
```

```
##                                    cluster_1
## D_5__Ruminiclostridium 5               TRUE
## D_5__Lachnoclostridium                 TRUE
## D_5__Holdemania                        TRUE
## D_5__Anaerostipes                      TRUE
## D_5__uncultured_3                      TRUE
## D_5__Ruminococcaceae NK4A214 group     TRUE
```

Let's collect information for the scatter plot.

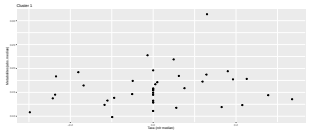
```
# Function for obtaining sample-wise sum, mean, median, and mean
 ↪ variance
# for each cluster
```

```
.sum_mean_median_var <- function(tse1, tse2, assay_name1,
 ↪  assay_name2, clusters1, clusters2){

  list <- list()
  # Create a data frame that includes all the information
  for(i in 1:ncol(clusters1) ){
    # Subset data based on cluster
    tse_subset1 <- tse1[clusters1[,i], ]
    tse_subset2 <- tse2[clusters2[,i], ]
    # Get assay
    assay1 <- assay(tse_subset1, assay_name1)
    assay2 <- assay(tse_subset2, assay_name2)
    # Calculate sum, mean, median, and mean variance
    sum1 <- colSums2(assay1, na.rm = T)
    mean1 <- colMeans2(assay1, na.rm = T)
    median1 <- colMedians(assay1, na.rm = T)
    var1 <- colVars(assay1, na.rm = T)

    sum2 <- colSums2(assay2, na.rm = T)
    mean2 <- colMeans2(assay2, na.rm = T)
    median2 <- colMedians(assay2, na.rm = T)
    var2 <- colVars(assay2, na.rm = T)

    list[[i]] <- data.frame(sample = colnames(tse1), sum1, sum2,
 ↪  mean1, mean2,
                      median1, median2, var1, var2)
  }

  return(list)
}

# Calculate info
df <- .sum_mean_median_var(mae[[1]], mae[[2]], "rclr", "nmr",
 ↪  bicluster_rows, bicluster_columns)
```

Now we can create a scatter plot. X-axis includes median clr abundance of microbiome and y-axis median absolute concentration of each metabolite. Each data point represents a single sample.

From the plots, we can see that there is low negative correlation in both cluster 1 and 3. This means that when abundance of bacteria belonging to cluster 1 or 3 is higher, the concentration of metabolites of cluster 1 or 3 is lower, and vice versa.

```
pics <- list()
for(i in seq_along(df)){
  pics[[i]] <- ggplot(df[[i]])  +
      geom_point(aes(x = median1, y = median2)) +
      labs(title = paste0("Cluster ", i),
           x = "Taxa (rclr median)",
           y = "Metabolites (abs. median)")
  print(pics[[i]])
}
```



```
# pics[[1]] + pics[[2]] + pics[[3]]
```

pheatmap does not allow boolean values, so they must be converted into factors.

```
bicluster_columns <- data.frame(apply(bicluster_columns, 2,
 ↪  as.factor))
bicluster_rows <- data.frame(apply(bicluster_rows, 2, as.factor))
```

Again, we can plot clusters with heatmap.

```
# Adjust colors for all clusters
if( ncol(bicluster_rows) > ncol(bicluster_columns) ){
  cluster_names <- colnames(bicluster_rows)
} else {
  cluster_names <- colnames(bicluster_columns)
}
annotation_colors <- list()
for(name in cluster_names){
  annotation_colors[[name]] <- c("TRUE" = "red", "FALSE" =
 ↪  "white")
}

# Create a heatmap
pheatmap(corr, cluster_cols = F, cluster_rows = F,
         annotation_col = bicluster_columns,
         annotation_row = bicluster_rows,
         annotation_colors = annotation_colors)
```

10.5.3 Taxa vs taxa

Third and final example deals with situation where we want to analyze correlation between taxa. *biclust* is suitable for this.

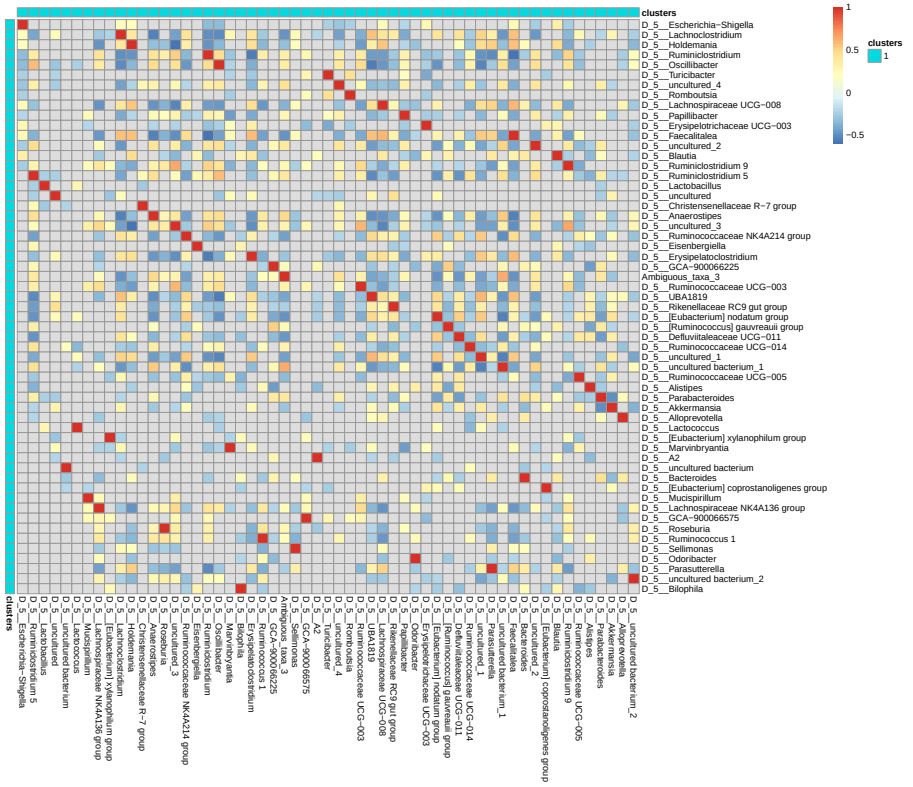
```
# Calculate cross-correlation
corr <- getExperimentCrossCorrelation(mae, 1, 1,
                                      assay_name1 = "rclr",
                                       ↪  assay_name2 = "rclr",
                                      mode = "matrix",
                                      cor_threshold = 0.2,
                                       ↪  verbose = F,
                                       ↪  show_warning = F)

# Find biclusters
```

```
bc <- biclust(corr, method=BCPlaid(), fit.model = y ~ m,
              background = TRUE, shuffle = 100, back.fit = 0,
               ↪  max.layers = 10,
              iter.startup = 10, iter.layer = 100, verbose =
               ↪  FALSE)
```

```
# Get biclusters
bcs <- .get_biclusters_from_biclust(bc, corr)

bicluster_rows <- bcs$bc_rows
bicluster_columns <- bcs$bc_columns
```

```
# Create a column that combines information
# If row/column includes in multiple clusters, cluster numbers
 ↪  are separated with "_&_"
bicluster_columns$clusters <- apply(bicluster_columns, 1,

                                    ↪  function(x){paste(paste(which(x)),
                                    ↪  collapse = "_&_") })
bicluster_columns <- bicluster_columns[, "clusters", drop =
 ↪  FALSE]

bicluster_rows$clusters <- apply(bicluster_rows, 1,

                                  ↪  function(x){paste(paste(which(x)),
                                  ↪  collapse = "_&_") })
bicluster_rows <- bicluster_rows[, "clusters", drop = FALSE]
```

```
# Convert boolean values into factor
bicluster_columns <- data.frame(apply(bicluster_columns, 2,
 ↪  as.factor))
bicluster_rows <- data.frame(apply(bicluster_rows, 2, as.factor))

pheatmap(corr, cluster_cols = F, cluster_rows = F,
         annotation_col = bicluster_columns,
         annotation_row = bicluster_rows)
```



10.6 Additional Community Typing

For more community typing techniques applied to the 'SprockettTHData' data set, see the attached .Rmd file.

Link:

- Rmd

Chapter 11

Differential abundance

11.1 Differential abundance analysis

This section provides an overview and examples of *differential abundance analysis (DAA)* based on one of the openly available datasets in mia to illustrate how to perform differential abundance analysis (DAA). DAA identifies differences in the abundances of individual taxonomic groups between two or more groups (e.g. treatment vs control). This can be performed at any phylogenetic level.

We perform DAA to identify biomarkers and/or gain understanding of a complex system by looking at its isolated components. For example, identifying that a bacterial taxon is different between a patient group with disease X vs a healthy control group might lead to important insights into the pathophysiology. Changes in the microbiota might be cause or a consequence of a disease. Either way, it can help to understand the system as a whole. Be aware that this approach has also been criticized recently (Quinn et al., 2021).

11.1.1 Examples and tools

There are many tools to perform DAA. The most popular tools, without going into evaluating whether or not they perform well for this task, are:

- ALDEx2 (Gloor et al., 2016)
- ANCOMBC (Lin and Peddada, 2020a)
- corncob (Martin et al., 2021)
- DESeq2 (Love et al., 2014)
- edgeR (Chen et al., 2016)
- lefser (Khleborodova, 2021)
- Maaslin2 (Mallick et al., 2020)

- metagenomeSeq (Paulson et al., 2017)
- limma (Ritchie et al., 2015)
- LinDA (Zhou et al., 2022)
- t-test

- Wilcoxon test

We recommend to have a look at Nearing et al. (2022) who compared all these listed methods across 38 different datasets. Because different methods use different approaches (parametric vs non-parametric, different normalization techiniques, assumptions etc.), results can differ between methods. Unfortunately, as Nearing et al. (2022) point out, they can differ substantially. Therefore, it is highly recommended to pick several methods to get an idea about how robust and potentially reproducible your findings are depending on the method. In this section we demonstrate 4 methods that can be recommended based on recent literature (ANCOM-BC, ALDEx2, Maaslin2 and LinDA) and we will compare the results between them. Note that the purpose of this section is to show how to perform DAA in R, not how to correctly do causal inference. Depending on your experimental setup and your theory, you must determine how to specify any model exactly. E.g., there might be confounding factors that might drive (the absence of) differences between the shown groups that we ignore here for simplicity. However, we will show how you could include covariates in those models. Furthermore, we picked a dataset that merely has microbial abundances in a TSE object as well as a grouping variable in the sample data. We simplify the analysis by only including 2 of the 3 groups.

```
library(mia)
library(patchwork)
library(tidySummarizedExperiment)
library(ALDEx2)
library(Maaslin2)
library(MicrobiomeStat)
library(knitr)
library(tidyverse)
library(ANCOMBC)

# set random seed because some tools can randomly vary and then
 ↪  produce
# different results:
set.seed(13253)

# we use a demo dataset and restrict it to two geo locations
# for easy illustration
data(peerj13075)
tse <- peerj13075
```

```
tse <- tse[ ,tse$Geographical_location %in% c("Pune", "Nashik")]
# Let us make this a factor
tse$Geographical_location <- factor(tse$Geographical_location)

# how many observations do we have per group?
count(as.data.frame(colData(tse)), Geographical_location) %>%
 ↪  kable()
```

Geographical_location	n
Nashik	11
Pune	36

11.1.2 Prevalence Filtering

Before we jump to our analyses, we may want to perform prevalence filtering. Nearing et al. (2022) found that applying a 10% threshold for the prevalence of the taxa generally resulted in more robust results. Some tools have builtin arguments for that. By applying the threshold to our input data, we can make sure it is applied for all tools. Below we show how to do this in `mia`:

```
tse <- subsetByPrevalentTaxa(tse, detection = 0, prevalence =
 ↪  0.1)
```

11.1.3 ALDEx2

In this section, we will show how to perform a simple ALDEx2 analysis. If you wanted to pick a single method, this method could be recommended to use. According to the developers experience, it tends to identify the common features identified by other methods. This statement is in line with a recent independent evaluation by Nearing et al. (2022).
Please also have a look at the more extensive vignette that covers this flexible tool in more depth. ALDEx2 estimates technical variation within each sample per taxon by utilizing the Dirichlet distribution. It furthermore applies the centered-log-ratio transformation (or closely related log-ratio transforms). Depending on the experimental setup, it will perform a two sample Welch's T-test and Wilcoxon-test or a one-way ANOVA and Kruskal-Wallis-test. For more complex study designs, there is a possibility to utilize the `glm` functionality within ALDEx2. The Benjamini-Hochberg procedure is applied in any case to correct for multiple testing. Below we show a simple example that illustrates the workflow.

```
# Generate Monte Carlo samples of the Dirichlet distribution for
 ↪  each sample.
# Convert each instance using the centered log-ratio transform.
# This is the input for all further analyses.
set.seed(254)
x <- aldex.clr(assay(tse), tse$Geographical_location)
```

The t-test:

```
# calculates expected values of the Welch's t-test and Wilcoxon
 ↪  rank
# test on the data returned by aldex.clr
x_tt <- aldex.ttest(x, paired.test = FALSE, verbose = FALSE)
```

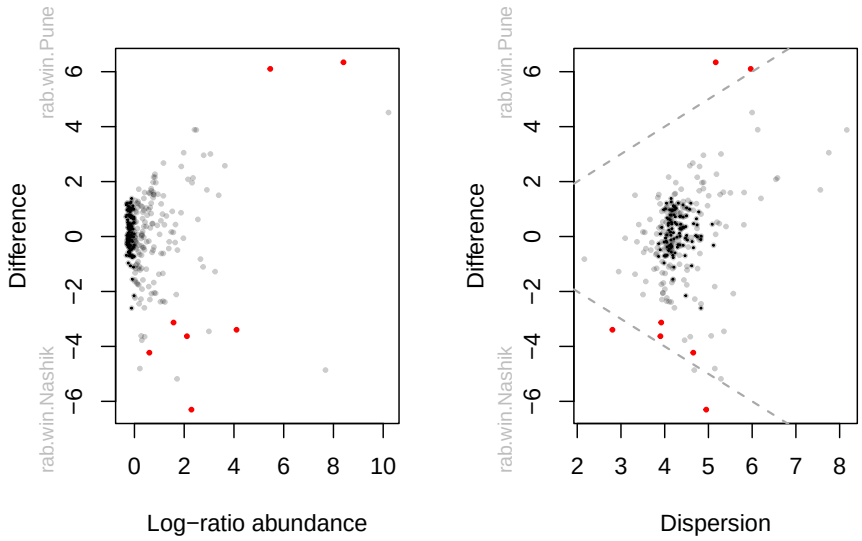
Effect sizes:

```
# determines the median clr abundance of the feature in all
 ↪  samples and in
# groups, the median difference between the two groups, the
 ↪  median variation
# within each group and the effect size, which is the median of
 ↪  the ratio
# of the between group difference and the larger of the variance
 ↪  within groups
x_effect <- aldex.effect(x, CI = TRUE, verbose = FALSE)

# combine all outputs
aldex_out <- data.frame(x_tt, x_effect)
```

Now, we can create a so called Bland-Altman or MA plot (left). It shows the association between the relative abundance and the magnitude of the difference per sample. Next to that, we can also create a plot that shows the dispersion on the x-axis instead of log-ratio abundance. Red dots represent genera that are differentially abundant ($q \leq 0.1$) between the 2 groups. Black points are rare taxa and grey ones are abundant taxa. The dashed line represent an effect size of 1. See Gloor et al. (2016) to learn more about these plots.

```
par(mfrow = c(1, 2))
  aldex.plot(
    aldex_out,
    type = "MA",
    test = "welch",
    xlab = "Log-ratio abundance",
```

```
    ylab = "Difference",
    cutoff = 0.05
  )
  aldex.plot(
    aldex_out,
    type = "MW",
    test = "welch",
    xlab = "Dispersion",
    ylab = "Difference",
    cutoff = 0.05
  )
```



The evaluation as differential abundant in above plots is based on the corrected p-value. According to the ALDEx2 developers, the safest approach is to identify those features where the 95% CI of the effect size does not cross 0. As we can see in below table, this is not the case for any of the identified genera (see overlap column, which indicates the proportion of overlap). Also, the authors recommend to focus on effect sizes and CIs rather than interpreting the p-value. To keep the comparison simple, we will here use the p-value as decision criterion. But please be aware that the effect size together with the CI is a better answer to the question we are typically interested in (see also this article).

```
rownames_to_column(aldex_out, "genus") %>%
  filter(wi.eBH <= 0.05)  %>% # here we chose the wilcoxon output
    ↪  rather than tt
```

```
  select(genus, we.eBH, wi.eBH, effect, overlap) %>%
  kable()
```

genus	we.eBH	wi.eBH	effect	overlap
OTU194	0.0553	0.0151	0.9552	0.1563
OTU562	0.0714	0.0266	-0.7147	0.1648
OTU611	0.1204	0.0468	-0.7630	0.1745
OTU773	0.0281	0.0036	1.1930	0.1037
OTU860	0.0866	0.0380	-0.8501	0.1733
OTU1075	0.0374	0.0083	-1.1059	0.1278
OTU1235	0.0280	0.0253	-0.9094	0.1702
OTU1680	0.0834	0.0341	-0.9270	0.1915
OTU2529	0.1113	0.0449	-0.8634	0.1929

11.1.4 ANCOM-BC

The analysis of composition of microbiomes with bias correction (ANCOM-BC) (Lin and Peddada, 2020b) is a recently developed method for differential abundance testing. It is based on an earlier published approach (Mandal et al., 2015). The previous version of ANCOM was among the methods that produced the most consistent results and is probably a conservative approach (Nearing et al., 2022). However, the new ANCOM-BC method operates quite differently compared to the former ANCOM method.

As the only method, ANCOM-BC incorporates the so called *sampling fraction* into the model. The latter term could be empirically estimated by the ratio of the library size to the microbial load. According to the authors, variations in this sampling fraction would bias differential abundance analyses if ignored. Furthermore, this method provides p-values and confidence intervals for each taxon. It also controls the FDR and it is computationally simple to implement.

Note that the original method was implemented in the `ancombc()` function (see extended tutorial). The method has since then been updated and new features have been added to enable multi-group comparisons and repeated measurements among other improvements. We do not cover the more advanced features of ANCOMBC in this tutorial as these features are documented in detail in this tutorial.

We now proceed with a simple example. First, we specify a formula. In this formula, other covariates could potentially be included to adjust for confounding. We show this further below. Again, please make sure to check the function documentation as well as the linked tutorials to learn about the additional arguments that we specify.

```
# perform the analysis
out <- ancombc2(
  data = tse,
  tax_level="genus",
  fix_formula = "Geographical_location",
  p_adj_method = "fdr",
  prv_cut = 0, # prev filtering has been done above already
  lib_cut = 0,
  group = "Geographical_location",
  struc_zero = TRUE,
  neg_lb = TRUE,
  iter_control = list(tol = 1e-5, max_iter = 20, verbose =
   ↪  FALSE),
  em_control = list(tol = 1e-5, max_iter = 20), # use max_iter >=
   ↪  100 on real data
  alpha = 0.05,
  global = TRUE # multi group comparison will be deactivated
   ↪  automatically
)
# store the results in res
res <- out$res
```

The object `out` contains all model output. Again, see the documentation of the function under **Value** for an explanation of all the output objects. Our question whether taxa are differentially abundant can be answered by looking at the `res` object, which now contains dataframes with the coefficients, standard errors, p-values and q-values. Conveniently, there is a dataframe `diff_abn`. Here, for each taxon it is indicated whether it is differentially abundant between the groups (again, keep in mind that the answer is not black-white). Below we show the first 6 entries of this dataframe:

```
kable(head(res))
```

taxon	lfc_(Intercept)	lfc_Geographical_locationPune	se_(Intercept)	se_Geographical_locatic
Abyssicoccus	0.0399	-0.0570	0.1675	
Acidaminococcus	0.6874	-0.9024	0.1947	
Acinetobacter	0.1243	-0.1672	0.7823	
Actinomyces	0.1347	-0.1807	0.1938	
Actinoplanes	0.2716	-0.3594	0.1635	
Aerococcus	0.0237	-0.0358	0.1677	

11.1.5 MaAsLin2

Next, we will illustrate how to use MaAsLin2, which is the next generation of MaAsLin. As it is based on generalized linear models, it is flexible for different study designs and covariate structures. The official package tutorial can be found here.

```
# maaslin expects features as columns and samples as rows
# for both the asv/otu table as well as meta data
asv <- t(assay(tse))
meta_data <- data.frame(colData(tse))
# you can specifiy different GLMs/normalizations/transforms. We
 ↪  used similar
# settings as in Nearing et al. (2021) here:
fit_data <- Maaslin2(
  asv,
  meta_data,
  output = "DAA example",
  transform = "AST",
  fixed_effects = "Geographical_location",
  # random_effects = c(...), # you can also fit MLM by specifying
   ↪  random effects
  # specifying a ref is especially important if you have more
   ↪  than 2 levels
  reference = "Geographical_location,Pune",
  normalization = "TSS",
  standardize = FALSE,
  min_prevalence = 0 # prev filterin already done
)
```

Which genera are identified as differentially abundant? (leave out "head" to see all).

```
kable(head(filter(fit_data$results, qval <= 0.05)))
```

feature	metadata	value	coef	stderr	pval	name
OTU1053	Geographical_location	Pune	-0.0080	0.0011	0	Geographical_locationPune
OTU860	Geographical_location	Pune	-0.0373	0.0059	0	Geographical_locationPune
OTU1075	Geographical_location	Pune	-0.1295	0.0207	0	Geographical_locationPune
OTU1980	Geographical_location	Pune	-0.0395	0.0062	0	Geographical_locationPune
OTU611	Geographical_location	Pune	-0.0274	0.0045	0	Geographical_locationPune
OTU2335	Geographical_location	Pune	-0.0089	0.0015	0	Geographical_locationPune

A folder will be created that is called like the above specified output. It contains also figures to visualize the difference between genera for the significant ones.

11.1.6 LinDA

Lastly, we cover linear models for differential abundance analysis of microbiome compositional data (Zhou et al. (2022)). This tool is very similar to ANCOMBC with few differences: 1) LinDA correct for the compositional bias differently using the mode of all regression coefficients. 2) The authors claim that it runs 100-1000x faster than ANCOMBC and 3) it support hierarchical models. The latter could be ignored as ANCOMBC will be supporting hierarchical models with the next release. Nevertheless, LinDA seems a promising tool that achieves the best power/fdr trade-off together with ANCOMBC according to the authors. The speed might make it the choice for bigger datasets or datasets with a very high number of features.

```
otu.tab <- as.data.frame(assay(tse))
meta <- as.data.frame(colData(tse)) %>%
 ↪  select(Geographical_location)
res <- linda(
  otu.tab,
  meta,
  formula = '~Geographical_location',
  alpha = 0.05,
  prev.filter = 0,
  mean.abund.filter = 0)
```

```
## 0  features are filtered!
## The filtered data has  47  samples and  262  features will be tested!
## Pseudo-count approach is used.
## Fit linear models ...
## Completed.
```

```
# to scan the table for genera where H0 could be rejected:
kable(head(filter(as.data.frame(res$output$Geographical_locationPune),
 ↪  reject)))
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj	reject	df
OTU15	1194.9	-1.9113	0.3579	-5.340	0.0000	0.0000	TRUE	45
OTU22	393.5	-0.6683	0.2184	-3.060	0.0037	0.0160	TRUE	45
OTU76	837.0	-1.8013	0.3598	-5.006	0.0000	0.0001	TRUE	45
OTU127	938.2	-1.7558	0.3912	-4.488	0.0000	0.0004	TRUE	45
OTU170	416.9	-0.7518	0.2351	-3.197	0.0025	0.0123	TRUE	45
OTU194	869.3	4.3671	1.2254	3.564	0.0009	0.0054	TRUE	45

11.1.7 Comparison of the methods

When we compare the methods in the context of a research question, we could look at e.g. at whether they agree based on the applied decision criterion (e.g. adjusted p value < 0.05). That is what we illustrate here. First we will look at how many taxa were identified by each method to begin with. In the next step we will look at the intersection of identified taxa. To achieve that, we first create a dataframe that summarises the decision criterion for each method and shows a score from 0 to 3 indicating how many methods agreed on a particular taxon.

```
# change genus names to otu ids for ancombc results to make it
 ↪  joinable with others
id_switch <- as.data.frame(rowData(tse)) %>%
  rownames_to_column("taxid") %>%
  select(taxid, genus)
abc_res <- select(out$res, genus = taxon, ancombc =
 ↪  diff_Geographical_locationPune) %>%
  left_join(id_switch, by = "genus") %>%
  select(-genus)

# join all results together
summ <- full_join(
    rownames_to_column(aldex_out, "taxid") %>%
      select(taxid, aldex2 = wi.eBH),
    abc_res,
    by = "taxid") %>%
  full_join(
    select(fit_data$results, taxid = feature, maaslin2 = qval),
    by = "taxid") %>%
    full_join(

       ↪  rownames_to_column(as.data.frame(res$output$Geographical_locationPune),
       ↪  "taxid") %>%
        select(taxid, LinDA = reject),
      by = "taxid") %>%
  mutate(
    across(c(aldex2, maaslin2), ~ .x <= 0.05),
    # the following line would be necessary without prevalence
     ↪  filtering
    # as some methods output NA
    #across(-genus, function(x) ifelse(is.na(x), FALSE, x)),
    ancombc = ifelse(is.na(ancombc), FALSE, ancombc),
    score = rowSums(across(c(aldex2, ancombc, maaslin2, LinDA))),
  )
```

```
# This is how it looks like:
kable(head(summ))
```

taxid	aldex2	ancombc	maaslin2	LinDA	score
OTU2	FALSE	FALSE	FALSE	FALSE	0
OTU15	FALSE	TRUE	TRUE	TRUE	3
OTU22	FALSE	FALSE	TRUE	TRUE	2
OTU53	FALSE	FALSE	FALSE	FALSE	0
OTU69	FALSE	FALSE	FALSE	FALSE	0
OTU76	FALSE	FALSE	TRUE	TRUE	2

Now we can answer our questions:

```
# how many genera were identified by each method?
summarise(summ, across(where(is.logical), sum)) %>%
  kable()
```

aldex2	ancombc	maaslin2	LinDA
9	31	67	75

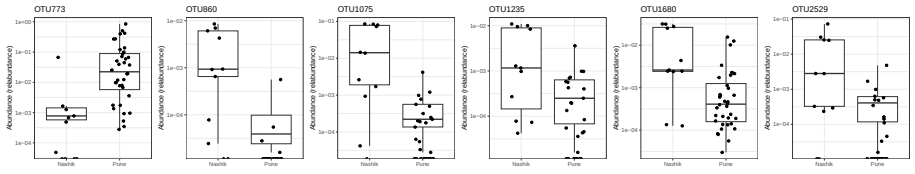
```
# which genera are identified by all methods?
filter(summ, score == 4) %>% kable()
```

taxid	aldex2	ancombc	maaslin2	LinDA	score
OTU773	TRUE	TRUE	TRUE	TRUE	4
OTU860	TRUE	TRUE	TRUE	TRUE	4
OTU1075	TRUE	TRUE	TRUE	TRUE	4
OTU1235	TRUE	TRUE	TRUE	TRUE	4
OTU1680	TRUE	TRUE	TRUE	TRUE	4
OTU2529	TRUE	TRUE	TRUE	TRUE	4

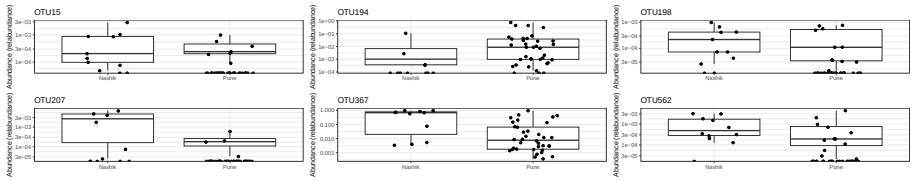
We see that each method identified at least some genera as differentially abundant. Many of those that were identified by ALDEx2, were also identified by the other methods. Let us plot the data for any method or for those taxa that were identified by all methods:

```
# Data
data(peerj13075)
tse <- peerj13075

# Add relative abundances and clr abundances
tse <- transformCounts(tse, method="relabundance")
tse <- transformCounts(tse, method="clr", pseudocount=1)
```

```
# Subset to prevalent taxa (exclude rare taxa at 10 percent
 ↪  prevalence using 0 detection threshold):
# do the subsetting based on the relative abundance assay
tse <- subsetByPrevalentTaxa(tse, detection = 0, prevalence =
 ↪  10/100, assay.type="relabundance")

# Subset to certain geolocations
tse <- tse[ ,tse$Geographical_location %in% c("Pune", "Nashik")]

# Let us make the geo location a factor
tse$Geographical_location <- factor(tse$Geographical_location)

# Create a jittered boxplot for each genus
assay.type <- "relabundance"
plot_data <- data.frame(t(assay(tse, assay.type)))
plot_data$Geographical_location <- tse$Geographical_location
plots <- pmap(select(summ, taxid, score), function(taxid, score)
 ↪  {
  ggplot(plot_data, aes_string(x="Geographical_location",
   ↪  y=taxid)) +
    geom_boxplot(outlier.shape = NA) +
    geom_jitter(width = 0.2) +
    scale_y_log10() +
    labs(title=glue::glue("{taxid}"), x="",
     ↪  y=glue::glue("Abundance ({assay.type})")) +
    theme_bw() +
    theme(legend.position = "none")
})

# now we can show only those genera that have at least score 3
 ↪  (or 2 or 1)
robust_plots <- plots[summ$score == 4 & !is.na(summ$score)]

# to display this nicely in the book we use patchwork here:
# (we show first ones)
robust_plots[[1]] +
  robust_plots[[2]] +
  robust_plots[[3]] +
  robust_plots[[4]] +
  robust_plots[[5]] +
  robust_plots[[6]] +
  plot_layout(nrow = 1)
```




```
# or if we have most trust in any specific method we can show
 ↪  genera that
# are differentially abundant according to that method and then
 ↪  look in the
# title how many methods also identified it (we only show first 6
 ↪  here):
ancombc_plots <- plots[summ$ancombc & !is.na(summ$score)]
ancombc_plots[[1]] +
  ancombc_plots[[2]] +
  ancombc_plots[[3]] +
  ancombc_plots[[4]] +
  ancombc_plots[[5]] +
  ancombc_plots[[6]]
```



11.1.8 Confounding variables

To perform causal inference, it is crucial that the method is able to include covariates in the model. This is not possible with e.g. the Wilcoxon test. Other methods such as both ANCOM methods, ALDEx2, LinDA, MaAsLin2 and others allow this. Below we show how to include a covariate in ANCOM-BC. It is very similar for all the methods that allow this. Since in this dataset there are no covariates, I first simulate a new variable and add it to the TSE object.

```
# FIXME: switch to a faster example / method
out_cov = ancombc2(
  data = tse,
  fix_formula = "Geographical_location + Age", # here we add Age
   ↪  to the model
  p_adj_method = "fdr",
  prv_cut = 0,  # we did that already
```

```
  lib_cut = 0,
  group = "Geographical_location",
  struc_zero = TRUE,
  neg_lb = TRUE,
  iter_control = list(tol = 1e-5, max_iter = 20, verbose =
   ↪  FALSE),
  em_control = list(tol = 1e-5, max_iter = 20),
  alpha = 0.05,
  global = TRUE # multi group comparison will be deactivated
   ↪  automatically
)

# now the model answers the question: holding Age constant, are
# bacterial taxa differentially abundant? Or, if that is of
 ↪  interest,
# holding phenotype constant, is Age associated with bacterial
 ↪  abundance?
# Again we only show the first 6 entries.
kable(head(out_cov$res))
```

taxon	lfc_(Intercept)	lfc_Geographical_locationPune	lfc_AgeElderly	lfc_AgeMiddle_age
OTU2	0.0441	-0.1005	0.0892	0.0118
OTU15	0.7071	-0.7615	-0.2435	-0.1584
OTU53	0.0248	-1.0822	1.4991	1.1526
OTU87	0.1808	0.1182	-0.4585	-0.4494
OTU99	0.3073	-0.1658	-0.2769	-0.3349
OTU111	0.0066	-0.1498	0.1351	0.2453

In the next section of this book chapter we cover methods that can also take into account the phylogenetic information of bacterial taxa to perform group-wise associations.

11.2 Tree-based methods

11.2.1 Group-wise associations testing based on balances

For testing associations based on balances, check the philr R/Bioconductor package.

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] doRNG_1.8.6                    rngtools_1.5.2
 [3] foreach_1.5.2                  ANCOMBC_2.1.3
 [5] lubridate_1.9.2                forcats_1.0.0
 [7] stringr_1.5.0                  dplyr_1.1.1
 [9] purrr_1.0.1                    readr_2.1.4
[11] tidyr_1.3.0                    tibble_3.2.1
[13] ggplot2_3.4.1                  tidyverse_2.0.0
[15] knitr_1.42                     MicrobiomeStat_1.1
[17] Maaslin2_1.10.0                ALDEx2_1.28.1
[19] zCompositions_1.4.0-1          truncnorm_1.0-9
[21] NADA_1.6-1.1                   survival_3.5-5
[23] MASS_7.3-58.3                  tidySummarizedExperiment_1.6.1
[25] patchwork_1.1.2                mia_1.7.11
[27] MultiAssayExperiment_1.24.0    TreeSummarizedExperiment_2.1.4
[29] Biostrings_2.66.0              XVector_0.38.0
[31] SingleCellExperiment_1.20.1    SummarizedExperiment_1.28.0
[33] Biobase_2.58.0                 GenomicRanges_1.50.2
[35] GenomeInfoDb_1.34.9            IRanges_2.32.0
[37] S4Vectors_0.36.2               BiocGenerics_0.44.0
[39] MatrixGenerics_1.10.0          matrixStats_0.63.0-9003
[41] BiocStyle_2.24.0               rebook_1.6.0

loaded via a namespace (and not attached):
  [1] estimability_1.4.1          coda_0.19-4
  [3] bit64_4.0.5                 multcomp_1.4-23
```

```
 [5] irlba_2.3.5.1               DelayedArray_0.24.0
 [7] data.table_1.14.8           rpart_4.1.19
 [9] doParallel_1.0.17           RCurl_1.98-1.12
[11] generics_0.1.3              ScaledMatrix_1.6.0
[13] TH.data_1.1-1               timeSeries_4021.105
[15] RSQLite_2.3.0               proxy_0.4-27
[17] bit_4.0.5                   tzdb_0.3.0
[19] DirichletMultinomial_1.40.0 viridis_0.6.2
[21] xfun_0.38                   fBasics_4022.94
[23] hms_1.1.3                   evaluate_0.20
[25] DEoptimR_1.0-11             fansi_1.0.4
[27] readxl_1.4.2                igraph_1.4.1
[29] DBI_1.1.3                   htmlwidgets_1.6.2
[31] hash_2.2.6.2                Rmpfr_0.9-1
[33] CVXR_1.0-11                 ellipsis_0.3.2
[35] energy_1.7-11               backports_1.4.1
[37] bookdown_0.33               permute_0.9-7
[39] sparseMatrixStats_1.10.0    vctrs_0.6.1
[41] cachem_1.0.7                withr_2.5.0
[43] robustbase_0.95-0           emmeans_1.8.5
[45] checkmate_2.1.0             vegan_2.6-4
[47] treeio_1.22.0               getopt_1.20.3
[49] cluster_2.1.4               gsl_2.1-8
[51] ape_5.7-1                   dir.expiry_1.4.0
[53] lazyeval_0.2.2              crayon_1.5.2
[55] labeling_0.4.2              pkgconfig_2.0.3
[57] nlme_3.1-162                vipor_0.4.5
[59] nnet_7.3-18                 rlang_1.1.0
[61] spatial_7.3-16              lifecycle_1.0.3
[63] sandwich_3.0-2              filelock_1.0.2
[65] phyloseq_1.40.0             rsvd_1.0.5
[67] cellranger_1.1.0            graph_1.74.0
[69] Matrix_1.5-3                lpsymphony_1.24.0
[71] zoo_1.8-11                  Rhdf5lib_1.18.2
[73] boot_1.3-28.1               base64enc_0.1-3
[75] beeswarm_0.4.0              viridisLite_0.4.1
[77] stabledist_0.7-1            rootSolve_1.8.2.3
[79] bitops_1.0-7                rhdf5filters_1.8.0
[81] blob_1.2.4                  DelayedMatrixStats_1.20.0
[83] decontam_1.18.0             DECIPHER_2.26.0
[85] beachmat_2.14.0             scales_1.2.1
[87] memoise_2.0.1               magrittr_2.0.3
[89] plyr_1.8.8                  zlibbioc_1.44.0
[91] compiler_4.2.1              RColorBrewer_1.1-3
[93] clue_0.3-64                 lme4_1.1-32
[95] cli_3.6.1                   ade4_1.7-22
```

```
 [97] lmerTest_3.1-3             pbapply_1.7-0
 [99] htmlTable_2.4.1            Formula_1.2-5
[101] mgcv_1.8-42                tidyselect_1.2.0
[103] stringi_1.7.12             highr_0.10
[105] yaml_2.3.7                 BiocSingular_1.14.0
[107] ggrepel_0.9.3              grid_4.2.1
[109] tools_4.2.1                lmom_2.9
[111] timechange_0.2.0           parallel_4.2.1
[113] rstudioapi_0.14            logging_0.10-108
[115] foreign_0.8-84             statip_0.2.3
[117] optparse_1.7.3             gridExtra_2.3
[119] gld_2.6.6                  farver_2.1.1
[121] stable_1.1.6               RcppZiggurat_0.1.6
[123] digest_0.6.31              BiocManager_1.30.20
[125] Rcpp_1.0.10                scuttle_1.8.4
[127] httr_1.4.5                 Rdpack_2.4
[129] colorspace_2.1-0           XML_3.99-0.14
[131] modeest_2.4.0              splines_4.2.1
[133] yulab.utils_0.0.6          rmutil_1.1.10
[135] statmod_1.5.0              tidytree_0.4.2
[137] expm_0.999-7               scater_1.26.1
[139] multtest_2.52.0            Exact_3.2
[141] plotly_4.10.1              xtable_1.8-4
[143] gmp_0.7-1                  jsonlite_1.8.4
[145] nloptr_2.0.3               CodeDepends_0.6.5
[147] timeDate_4022.108          Rfast_2.0.7
[149] R6_2.5.1                   Hmisc_5.0-1
[151] pillar_1.9.0               htmltools_0.5.5
[153] glue_1.6.2                 fastmap_1.1.1
[155] minqa_1.2.5                BiocParallel_1.32.6
[157] BiocNeighbors_1.16.0       class_7.3-21
[159] codetools_0.2-19           pcaPP_2.0-3
[161] mvtnorm_1.1-3              utf8_1.2.3
[163] lattice_0.20-45            numDeriv_2016.8-1.1
[165] ggbeeswarm_0.7.1           DescTools_0.99.48
[167] biglm_0.9-2.1              rmarkdown_2.21
[169] biomformat_1.24.0          munsell_0.5.0
[171] e1071_1.7-13               rhdf5_2.40.0
[173] GenomeInfoDbData_1.2.9     iterators_1.0.14
[175] reshape2_1.4.4             gtable_0.3.3
[177] rbibutils_2.2.13
```

Chapter 12

Machine learning

Machine learning (ML) is a part of artificial intelligence. There are multiple definitions, but "machine" refers to computation and "learning" to improving performance based on the data by finding patterns from it. Machine learning includes wide variety of methods from simple statistical methods to more complex methods such as neural-networks.

Machine learning can be divided into supervised and unsupervised machine learning. Supervised ML is used to predict outcome based on the data. Unsupervised ML is used, for example, to reduce dimensionality (e.g. PCA) and to find clusters from the data (e.g., k-means clustering).

12.1 Supervised machine learning

"Supervised" means that the training data is introduced before. The training data contains labels (e.g., patient status), and the model is fitted based on the training data. After fitting, the model is utilized to predict labels of data whose labels are not known.

```
library(mia)

# Load experimental data
data(peerj13075)
(tse <- peerj13075)
```

```
## class: TreeSummarizedExperiment
## dim: 674 58
## metadata(0):
```

```
## assays(1): counts
## rownames(674): OTU1 OTU2 ... OTU2567 OTU2569
## rowData names(6): kingdom phylum ... family genus
## colnames(58): ID1 ID2 ... ID57 ID58
## colData names(5): Sample Geographical_location Gender Age Diet
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

Let's first preprocess the data.

```
# Agglomerate data
tse <- agglomerateByRank(tse, rank = "order")

# Apply CLR transform
assay(tse, "pseudo") <- assay(tse, "counts") + 1
tse <- transformCounts(tse, assay_name = "pseudo", method =
 ↪  "relabundance")
tse <- transformCounts(tse, assay_name = "relabundance", method =
 ↪  "clr")

# Get assay
assay <- assay(tse, "clr")
# Transpose assay
assay <- t(assay)

# Convert into data.frame
df <- as.data.frame(assay)

# Add labels to assay
labels <- colData(tse)$Diet
labels <- as.factor(labels)
df$diet <- labels

df[5, 5]
```

```
## [1] -0.4612
```

In the example below, we use mikropml package. We try to predict the diet type based on the data.

```
if( !require("mikropml") ){
    install.packages("mikropml")
    library(mikropml)
}

# Run random forest
results <- run_ml(df, "rf", outcome_colname = 'diet',
                  kfold = 2, cv_times = 5, training_frac = 0.8)

# Print result
confusionMatrix(data =
 ↪  results$trained_model$finalModel$predicted,
                reference = results$trained_model$finalModel$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Mixed Veg
##      Mixed    14   8
##      Veg       9  16
##
##                Accuracy : 0.638
##                  95% CI : (0.485, 0.773)
##     No Information Rate : 0.511
##     P-Value [Acc > NIR] : 0.0536
##
##                   Kappa : 0.276
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.609
##             Specificity : 0.667
##          Pos Pred Value : 0.636
##          Neg Pred Value : 0.640
##              Prevalence : 0.489
##          Detection Rate : 0.298
##    Detection Prevalence : 0.468
##       Balanced Accuracy : 0.638
##
##        'Positive' Class : Mixed
##
```

mikropml offers easier interface to caret package. However, we can also use it directly.

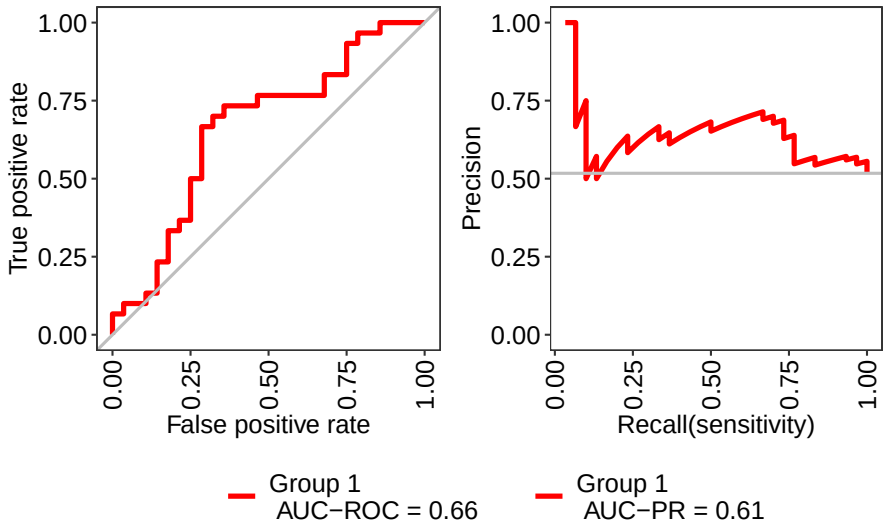
Let's use xgboost model which is another commonly used algorithm in bioinformatics.

```
# Set seed for reproducibility
set.seed(6358)

# Specify train control
train_control <- trainControl(method = "cv", number = 5,
                              classProbs = TRUE,
                              savePredictions = "final",
                              allowParallel = TRUE)

# Specify hyperparameter tuning grid
tune_grid <- expand.grid(nrounds = c(50, 100, 200),
                         max_depth = c(6, 8, 10),
                         colsample_bytree = c(0.6, 0.8, 1),
                         eta = c(0.1, 0.3),
                         gamma = 0,
                         min_child_weight = c(3, 4, 5),
                         subsample = c(0.6, 0.8)
                         )

# Train the model, use LOOCV to evaluate performance
model <- train(x = assay,
               y = labels,
               method = "xgbTree",
               objective = "binary:logistic",
               trControl = train_control,
               tuneGrid = tune_grid,
               metric = "AUC",
               verbosity = 0
)
```

Let's create ROC curve which is a commonly used method in binary classification. For unbalanced data, you might want to plot precision-recall curve.

```
if( !require(MLeval) ){
    install.packages("MLeval")
    library(MLeval)
}
# Calculate different evaluation metrics
res <- evalm(model, showplots = FALSE)

# Use patchwork to plot ROC and precision-recall curve
 ↪  side-by-side
```

```
library(patchwork)
res$roc + res$proc +
    plot_layout(guides = "collect") & theme(legend.position =
     ↪  'bottom')
```



12.2 Unsupervised machine learning

"Unsupervised" means that the labels (e.g., patient status is not known), and patterns are learned based only the abundance table, for instance. Unsupervised ML is also known as a data mining where patterns are extracted from big datasets.

For unsupervised machine learning, please refer to chapters that are listed below:

- Chapter 10
- Chapter 8

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] patchwork_1.1.2                MLeval_0.3
 [3] caret_6.0-94                   lattice_0.20-45
 [5] ggplot2_3.4.1                  mikropml_1.5.0
 [7] mia_1.7.11                     MultiAssayExperiment_1.24.0
 [9] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
[11] XVector_0.38.0                 SingleCellExperiment_1.20.1
[13] SummarizedExperiment_1.28.0    Biobase_2.58.0
[15] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[17] IRanges_2.32.0                 S4Vectors_0.36.2
[19] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[21] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[23] rebook_1.6.0

loaded via a namespace (and not attached):
  [1] plyr_1.8.8                  lazyeval_0.2.2
  [3] splines_4.2.1               BiocParallel_1.32.6
  [5] listenv_0.9.0               scater_1.26.1
  [7] digest_0.6.31               foreach_1.5.2
  [9] yulab.utils_0.0.6           htmltools_0.5.5
 [11] viridis_0.6.2               fansi_1.0.4
 [13] magrittr_2.0.3              memoise_2.0.1
 [15] MLmetrics_1.1.1             ScaledMatrix_1.6.0
 [17] cluster_2.1.4               ROCR_1.0-11
 [19] DECIPHER_2.26.0             recipes_1.0.5
 [21] globals_0.16.2              gower_1.0.1
```

```
 [23] hardhat_1.2.0               timechange_0.2.0
 [25] colorspace_2.1-0            blob_1.2.4
 [27] ggrepel_0.9.3               xfun_0.38
 [29] dplyr_1.1.1                 crayon_1.5.2
 [31] RCurl_1.98-1.12             jsonlite_1.8.4
 [33] graph_1.74.0                survival_3.5-5
 [35] iterators_1.0.14            ape_5.7-1
 [37] glue_1.6.2                  gtable_0.3.3
 [39] ipred_0.9-14                zlibbioc_1.44.0
 [41] DelayedArray_0.24.0         kernlab_0.9-32
 [43] BiocSingular_1.14.0         shape_1.4.6
 [45] future.apply_1.10.0         scales_1.2.1
 [47] DBI_1.1.3                   Rcpp_1.0.10
 [49] viridisLite_0.4.1           decontam_1.18.0
 [51] tidytree_0.4.2              proxy_0.4-27
 [53] bit_4.0.5                   rsvd_1.0.5
 [55] lava_1.7.2.1                prodlim_2019.11.13
 [57] glmnet_4.1-7                dir.expiry_1.4.0
 [59] farver_2.1.1                pkgconfig_2.0.3
 [61] XML_3.99-0.14               scuttle_1.8.4
 [63] nnet_7.3-18                 CodeDepends_0.6.5
 [65] utf8_1.2.3                  labeling_0.4.2
 [67] tidyselect_1.2.0            rlang_1.1.0
 [69] reshape2_1.4.4              munsell_0.5.0
 [71] tools_4.2.1                 cachem_1.0.7
 [73] xgboost_1.7.3.1             cli_3.6.1
 [75] DirichletMultinomial_1.40.0 generics_0.1.3
 [77] RSQLite_2.3.0               evaluate_0.20
 [79] stringr_1.5.0               fastmap_1.1.1
 [81] yaml_2.3.7                  ModelMetrics_1.2.2.2
 [83] knitr_1.42                  bit64_4.0.5
 [85] randomForest_4.7-1.1        purrr_1.0.1
 [87] future_1.32.0               nlme_3.1-162
 [89] sparseMatrixStats_1.10.0    compiler_4.2.1
 [91] beeswarm_0.4.0              filelock_1.0.2
 [93] e1071_1.7-13                treeio_1.22.0
 [95] tibble_3.2.1                stringi_1.7.12
 [97] highr_0.10                  Matrix_1.5-3
 [99] vegan_2.6-4                 permute_0.9-7
[101] vctrs_0.6.1                 pillar_1.9.0
[103] lifecycle_1.0.3             BiocManager_1.30.20
[105] BiocNeighbors_1.16.0        data.table_1.14.8
[107] bitops_1.0-7                irlba_2.3.5.1
[109] R6_2.5.1                    bookdown_0.33
[111] gridExtra_2.3               vipor_0.4.5
[113] parallelly_1.35.0           codetools_0.2-19
```

```
[115] MASS_7.3-58.3               withr_2.5.0
[117] GenomeInfoDbData_1.2.9      mgcv_1.8-42
[119] parallel_4.2.1              grid_4.2.1
[121] rpart_4.1.19                beachmat_2.14.0
[123] timeDate_4022.108           tidyr_1.3.0
[125] class_7.3-21                rmarkdown_2.21
[127] DelayedMatrixStats_1.20.0   pROC_1.18.0
[129] lubridate_1.9.2             ggbeeswarm_0.7.1
```

Chapter 13

Multi-assay analyses

```
library(mia)
```

Multi-omics means that we integrate data from multiple sources. For example, we can integrate microbial abundances in the gut with biomolecular profiling data from blood samples. This kind of integrative multi-omic approaches can support the analysis of microbiome dysbiosis and facilitate the discovery of novel biomarkers for health and disease.

With cross-correlation analysis, we can analyze how strongly and how differently variables are associated between each other. For instance, we can analyze if higher presence of a specific taxon equals to higher levels of a biomolecule.

The data containers that the *miaverse* utilizes are scalable and they can contain different types of data in a same container. Because of that, the *miaverse* is well-suitable for multi-assay microbiome data which incorporates different types of complementary data sources in a single reproducible workflow.

Another experiment can be stored in altExp slot of SE data container or both experiments can be stored side-by-side in MAE data container (see the sections 2.2.5 and 2.2.6 to learn more about altExp and MAE objects, respectively).

Different experiments are first imported into SE or TreeSE data container similarly to the case when only one experiment is present. After that different experiments are combined into the same data container. Result is one TreeSE object with alternative experiment in altExp slot, or MAE object with multiple experiment in its experiment slot.

As an example data, we use data from following publication: Hintikka L *et al.* (2021) Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: associating taxonomic and metabolomic patterns in fecal microbiotas with biclustering (Hintikka et al., 2021).

In this article, mice were fed with high-fat and low-fat diets with or without prebiotics. The purpose of this was to study if prebiotics would reduce the negative impacts of high-fat diet.

This example data can be loaded from microbiomeDataSets. The data is already in MAE format. It includes three different experiments: microbial abundance data, metabolite concentrations, and data about different biomarkers. Help for importing data into SE object you can find from here.

```
# Load the data
data(HintikkaXOData, package = "mia")
mae <- HintikkaXOData
mae
```

```
## A MultiAssayExperiment object of 3 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 3:
##  [1] microbiota: TreeSummarizedExperiment with 12706 rows and 40 columns
##  [2] metabolites: TreeSummarizedExperiment with 38 rows and 40 columns
##  [3] biomarkers: TreeSummarizedExperiment with 39 rows and 40 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

```
if(!require(stringr)){
    install.packages("stringr")
    library(stringr)
}
# Drop off those bacteria that do not include information in
 ↪  Phylum or lower levels
mae[[1]] <- mae[[1]][!is.na(rowData(mae[[1]])$Phylum), ]
# Clean taxonomy data, so that names do not include additional
 ↪  characters
rowData(mae[[1]]) <- DataFrame(apply(rowData(mae[[1]]), 2,
                                     str_remove, pattern =
 ↪  "._[0-9]__"))
# Microbiome data
mae[[1]]
```

```
## class: TreeSummarizedExperiment
```

```
## dim: 12613 40
## metadata(0):
## assays(1): counts
## rownames(12613): GAYR01026362.62.2014 CVJT01000011.50.2173 ...
##   JRJTB:03787:02429 JRJTB:03787:02478
## rowData names(7): Phylum Class ... Species OTU
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

```
# Metabolite data
mae[[2]]
```

```
## class: TreeSummarizedExperiment
## dim: 38 40
## metadata(0):
## assays(1): nmr
## rownames(38): Butyrate Acetate ... Malonate 1,3-dihydroxyacetone
## rowData names(0):
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

```
# Biomarker data
mae[[3]]
```

```
## class: TreeSummarizedExperiment
## dim: 39 40
## metadata(0):
## assays(1): signals
## rownames(39): Triglycerides_liver CLSs_epi ... NPY_serum Glycogen_liver
## rowData names(0):
```

```
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

13.1 Cross-correlation Analysis

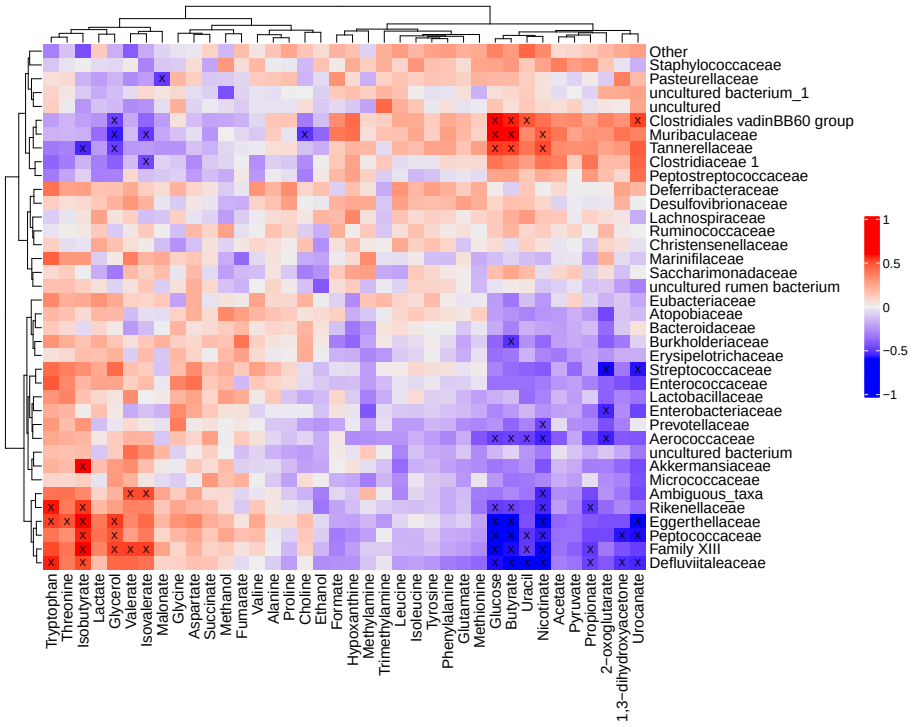
Next we can do the cross-correlation analysis. Here we analyse if individual bacteria genera correlates with concentrations of individual metabolites. This helps as to answer the question: "If this bacteria is present, is this metabolite's concentration then low or high"?

```
# Agglomerate microbiome data at family level
mae[[1]] <- agglomerateByPrevalence(mae[[1]], rank = "Family")
# Does log10 transform for microbiome data
mae[[1]] <- transformCounts(mae[[1]], method = "log10",
 ↪  pseudocount = 1)

# Give unique names so that we do not have problems when we are
 ↪  creating a plot
rownames(mae[[1]]) <- getTaxonomyLabels(mae[[1]])

# Cross correlates data sets
correlations <- testExperimentCrossCorrelation(mae,
                                               experiment1 = 1,
                                               experiment2 = 2,
                                               assay_name1 =
                                                ↪  "log10",
                                               assay_name2 =
                                                ↪  "nmr",
                                               method =
                                                ↪  "spearman",
                                               p_adj_threshold =
                                                ↪  NULL,
                                               cor_threshold =
                                                ↪  NULL,
                                               # Remove when mia
                                                ↪  is fixed
                                               mode = "matrix",
```

```
                                                    sort = TRUE,
                                                    show_warnings =
                                                     ↪ FALSE)
```

Creates the heatmap

```
if( !require("ComplexHeatmap") ){
    BiocManager::install("ComplexHeatmap")
    library("ComplexHeatmap")
}

# Create a heatmap and store it
plot <- Heatmap(correlations$cor,
                # Print values to cells
                cell_fun = function(j, i, x, y, width, height,
                 ↪ fill) {
                    # If the p-value is under threshold
                    if( !is.na(correlations$p_adj[i, j]) &
                     ↪ correlations$p_adj[i, j] < 0.05 ){
                        # Print "X"
                        grid.text(sprintf("%s", "X"), x, y, gp =
                         ↪ gpar(fontsize = 8, col = "black"))
                        }
                    },
                heatmap_legend_param = list(title = "",
                 ↪ legend_height = unit(5, "cm"))
                )
plot
```



13.2 Multi-Omics Factor Analysis

Multi-Omics Factor Analysis (Argelaguet, 2018) (MOFA) is an unsupervised method for integrating multi-omic data sets in a downstream analysis. It could be seen as a generalization of principal component analysis. Yet, with the ability to infer a latent (low-dimensional) representation, shared among the mutliple (-omics) data sets in hand.

We use the R MOFA2 package for the analysis, and install the corresponding dependencies.

```
if(!require(MOFA2)){
    BiocManager::install("MOFA2")
}

# For inter-operability between Python and R, and setting Python
 ↪  dependencies,
# reticulate package is needed
if(!require(reticulate)){
    install.packages("reticulate")
```

```
}

# Let us assume that these have been installed already.
#reticulate::install_miniconda(force = TRUE)
#reticulate::use_miniconda(condaenv = "env1", required = FALSE)
#reticulate::py_install(packages = c("mofapy2"), pip = TRUE,
 ↪  python_version=3.6)
```

The `mae` object could be used straight to create the MOFA model. Yet, we transform our assays since the model assumes normality per default. Other distributions that can be used, include Poisson or Bernoulli.

```
library(MOFA2)
# For simplicity, classify all high-fat diets as high-fat, and
 ↪  all the low-fat
# diets as low-fat diets
colData(mae)$Diet <- ifelse(colData(mae)$Diet == "High-fat" |
                              colData(mae)$Diet == "High-fat +
                               ↪  XOS",
                            "High-fat", "Low-fat")

# Removing duplicates at the microbiome data
# which are also in form e.g. "Ambiguous" and "uncultured" taxa
mae[[1]] <- mae[[1]][!duplicated(rownames(assay(mae[[1]]))), ]

# Transforming microbiome data with rclr
mae[[1]] <- transformCounts(mae[[1]], method = "relabundance")
mae[[1]] <- transformCounts(mae[[1]], assay_name =
 ↪  "relabundance", method = "rclr")

# Transforming metabolomic data with log10
mae[[2]] <- transformCounts(mae[[2]], assay_name = "nmr",
                            MARGIN = "samples",
                            method = "log10")

# Transforming biomarker data with z-transform
mae[[3]] <- transformCounts(mae[[3]], assay_name = "signals",
                            MARGIN = "features",
                            method = "z", pseudocount = 1)

# Removing assays no longer needed
assay(mae[[1]], "counts") <- NULL
assay(mae[[1]], "log10") <- NULL
assay(mae[[2]], "nmr") <- NULL
assay(mae[[3]], "signals") <- NULL
```

```
# Building our mofa model
model <- create_mofa_from_MultiAssayExperiment(mae,
                                               groups = "Diet",
                                               extract_metadata =
                                                ↪ TRUE)
model
```

```
## Untrained MOFA model with the following characteristics:
##  Number of views: 3
##  Views names: microbiota metabolites biomarkers
##  Number of features (per view): 38 38 39
##  Number of groups: 2
##  Groups names: High-fat Low-fat
##  Number of samples (per group): 20 20
##
```

Model options could be defined as follows:

```
model_opts <- get_default_model_options(model)
model_opts$num_factors <- 5
head(model_opts)
```

```
## $likelihoods
##  microbiota metabolites  biomarkers
##  "gaussian"  "gaussian"  "gaussian"
##
## $num_factors
## [1] 5
##
## $spikeslab_factors
## [1] FALSE
##
## $spikeslab_weights
## [1] TRUE
##
## $ard_factors
## [1] TRUE
##
## $ard_weights
## [1] TRUE
```

Model's training options are defined with the following:

```
train_opts <- get_default_training_options(model)
head(train_opts)
```

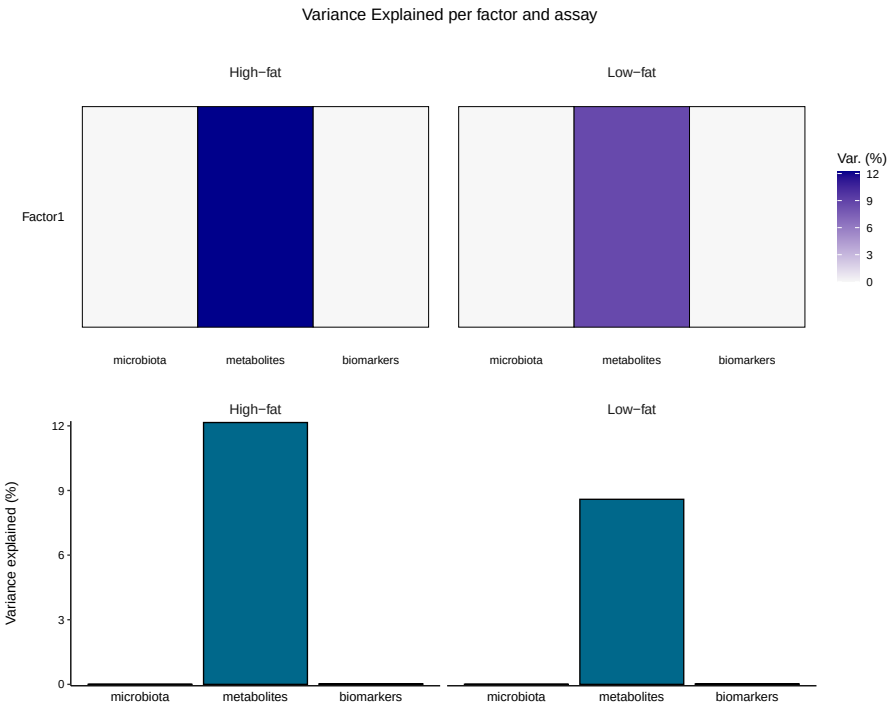
```
## $maxiter
## [1] 1000
##
## $convergence_mode
## [1] "fast"
##
## $drop_factor_threshold
## [1] -1
##
## $verbose
## [1] FALSE
##
## $startELBO
## [1] 1
##
## $freqELBO
## [1] 5
```

Preparing and training the model:

```
model.prepared <- prepare_mofa(
  object = model,
  model_options = model_opts
)

#Some systems may need the `use_basilisk = TRUE` function
#so it has been added to the following code
model.trained <- run_mofa(model.prepared, use_basilisk = TRUE)
```

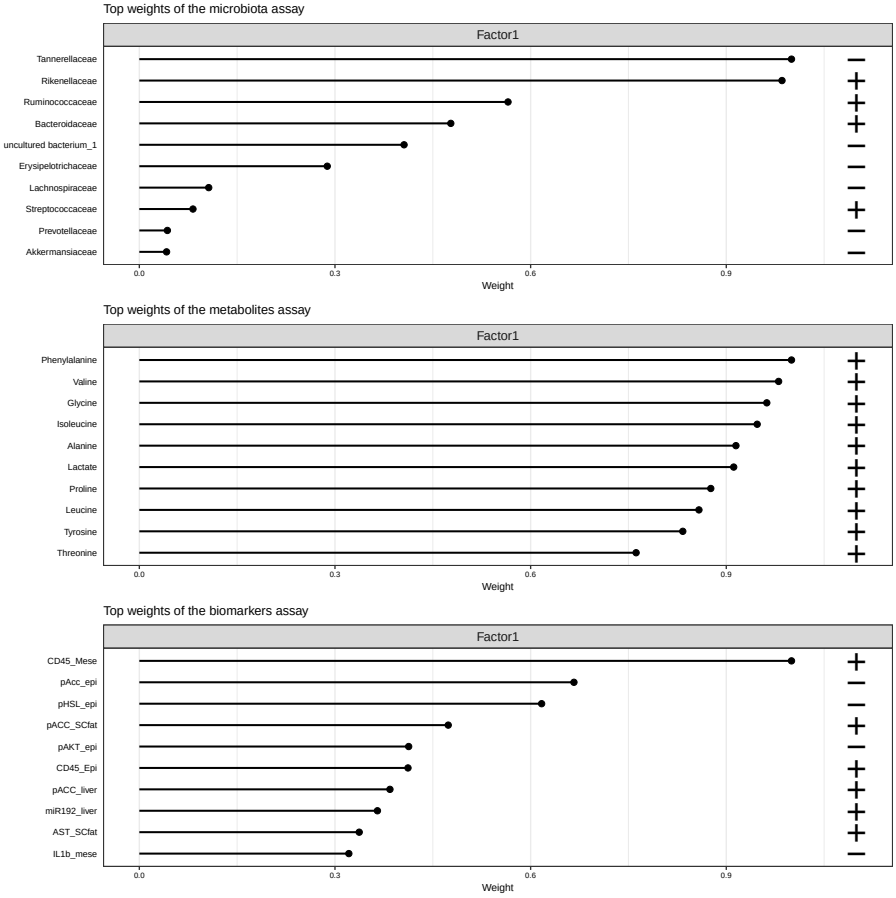
Visualizing the variance explained:

```
library(patchwork)
library(ggplot2)
wrap_plots(
    plot_variance_explained(model.trained, x="view", y="factor",
     ↪  plot_total = T),
    nrow = 2
) + plot_annotation(title = "Variance Explained per factor and
 ↪  assay",
                    theme = theme(plot.title = element_text(hjust
                     ↪  = 0.5)))
```



The top weights for each assay using all 5 factors:

```
plots <- lapply(c("microbiota", "metabolites","biomarkers"),
 ↪  function(name) {
    plot_top_weights(model.trained,
                     view = name,
                     factors = "all",
                     nfeatures = 10) +
        labs(title = paste0("Top weights of the ", name,"
         ↪  assay"))
})
wrap_plots(plots, nrow = 3) & theme(text = element_text(size =
 ↪  8))
```



More tutorials and examples of using the package are found at: link

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
```

```
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] grid      stats4    stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] ggplot2_3.4.1                  patchwork_1.1.2
 [3] reticulate_1.28                MOFA2_1.6.0
 [5] ComplexHeatmap_2.12.1          stringr_1.5.0
 [7] mia_1.7.11                     MultiAssayExperiment_1.24.0
 [9] TreeSummarizedExperiment_2.1.4 Biostrings_2.66.0
[11] XVector_0.38.0                 SingleCellExperiment_1.20.1
[13] SummarizedExperiment_1.28.0    Biobase_2.58.0
[15] GenomicRanges_1.50.2           GenomeInfoDb_1.34.9
[17] IRanges_2.32.0                 S4Vectors_0.36.2
[19] BiocGenerics_0.44.0            MatrixGenerics_1.10.0
[21] matrixStats_0.63.0-9003        BiocStyle_2.24.0
[23] rebook_1.6.0

loaded via a namespace (and not attached):
  [1] circlize_0.4.15             corrplot_0.92
  [3] BiocBaseUtils_1.0.0         plyr_1.8.8
  [5] lazyeval_0.2.2              splines_4.2.1
  [7] BiocParallel_1.32.6         scater_1.26.1
  [9] digest_0.6.31               foreach_1.5.2
 [11] yulab.utils_0.0.6           htmltools_0.5.5
 [13] viridis_0.6.2               fansi_1.0.4
 [15] magrittr_2.0.3              memoise_2.0.1
 [17] ScaledMatrix_1.6.0          cluster_2.1.4
 [19] doParallel_1.0.17           DECIPHER_2.26.0
 [21] colorspace_2.1-0            blob_1.2.4
 [23] ggrepel_0.9.3               xfun_0.38
 [25] dplyr_1.1.1                 crayon_1.5.2
 [27] RCurl_1.98-1.12             jsonlite_1.8.4
 [29] graph_1.74.0                iterators_1.0.14
 [31] ape_5.7-1                   glue_1.6.2
 [33] gtable_0.3.3                zlibbioc_1.44.0
 [35] GetoptLong_1.0.5            DelayedArray_0.24.0
 [37] BiocSingular_1.14.0         Rhdf5lib_1.18.2
 [39] shape_1.4.6                 HDF5Array_1.24.2
```

```
 [41] scales_1.2.1                pheatmap_1.0.12
 [43] DBI_1.1.3                   Rcpp_1.0.10
 [45] viridisLite_0.4.1           decontam_1.18.0
 [47] clue_0.3-64                 tidytree_0.4.2
 [49] bit_4.0.5                   rsvd_1.0.5
 [51] dir.expiry_1.4.0            RColorBrewer_1.1-3
 [53] farver_2.1.1                pkgconfig_2.0.3
 [55] XML_3.99-0.14               scuttle_1.8.4
 [57] uwot_0.1.14                 CodeDepends_0.6.5
 [59] here_1.0.1                  utf8_1.2.3
 [61] labeling_0.4.2              tidyselect_1.2.0
 [63] rlang_1.1.0                 reshape2_1.4.4
 [65] munsell_0.5.0               tools_4.2.1
 [67] cachem_1.0.7                cli_3.6.1
 [69] DirichletMultinomial_1.40.0 generics_0.1.3
 [71] RSQLite_2.3.0               evaluate_0.20
 [73] fastmap_1.1.1               yaml_2.3.7
 [75] knitr_1.42                  bit64_4.0.5
 [77] purrr_1.0.1                 nlme_3.1-162
 [79] sparseMatrixStats_1.10.0    compiler_4.2.1
 [81] beeswarm_0.4.0              filelock_1.0.2
 [83] png_0.1-8                   treeio_1.22.0
 [85] tibble_3.2.1                stringi_1.7.12
 [87] highr_0.10                  basilisk.utils_1.8.0
 [89] forcats_1.0.0               lattice_0.20-45
 [91] Matrix_1.5-3                vegan_2.6-4
 [93] permute_0.9-7               vctrs_0.6.1
 [95] pillar_1.9.0                lifecycle_1.0.3
 [97] rhdf5filters_1.8.0          BiocManager_1.30.20
 [99] GlobalOptions_0.1.2         BiocNeighbors_1.16.0
[101] cowplot_1.1.1               bitops_1.0-7
[103] irlba_2.3.5.1               R6_2.5.1
[105] bookdown_0.33               gridExtra_2.3
[107] vipor_0.4.5                 codetools_0.2-19
[109] MASS_7.3-58.3               rhdf5_2.40.0
[111] rprojroot_2.0.3             rjson_0.2.21
[113] withr_2.5.0                 GenomeInfoDbData_1.2.9
[115] mgcv_1.8-42                 parallel_4.2.1
[117] beachmat_2.14.0             tidyr_1.3.0
[119] basilisk_1.8.1              rmarkdown_2.21
[121] DelayedMatrixStats_1.20.0   Rtsne_0.16
[123] Cairo_1.6-0                 ggbeeswarm_0.7.1
```

Chapter 14

Visualization

Whether a data set contains information on a microbial community or it originates from a different source, the way that data are visualized inevitably shapes how they will be interpreted, and motivates the next steps of the analysis.

A large variety of graphing methods belong to microbial analysis, but only few are the choices that will return useful answers about the data. Therefore, knowledge on the available tools and their possible applications plays an important role in selecting the most suitable method for the asked question.

This chapter introduces the reader to a number of visualization techniques found in this book, such as:

- bar plots
- box plots
- heatmaps
- ordination charts
- regression charts
- trees

The toolkit which provides the essential plotting functionality includes the following packages:

- *patchwork*, *cowplot*, *ggpubr* and *gridExtra*: plot layout and multi-panel plotting
- *miaViz*: specific visualization tools for `TreeSummaizedExperiment` objects
- *scater*: specific visualization tools for `SingleCellExperiment` objects
- *ggplot2*, *pheatmap*, *ggtree*, *sechm*: composition heatmaps
- *ANCOMBC*, *ALDEx2* and *Maaslin2*: visual differential abundance
- *fido*: tree-based methods for differential abundance
- *plotly*: animated and 3D plotting

For systematic and extensive tutorials on the visual tools available in *mia*, readers can refer to the following material:

- microbiome tutorials

14.1 Pre-analysis exploration

14.1.1 Accessing row and column data

`SCE` and `TreeSE` objects contain multiple layers of information in the form of rows, columns and meta data. The *scater* package supports in accessing, modifying and graphing the meta data related to features as well as samples.

```
# list row meta data
names(rowData(tse))
```

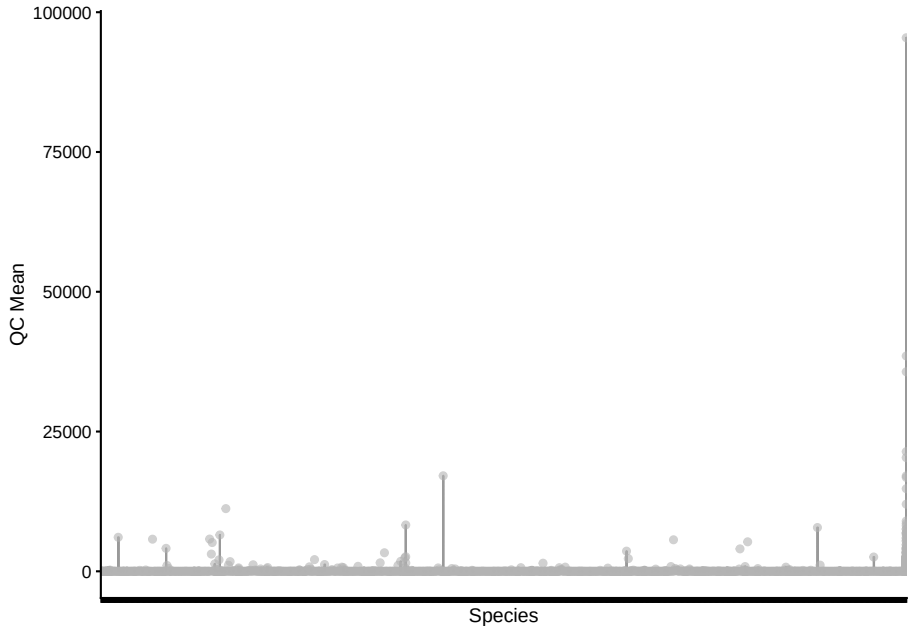
```
## [1] "Kingdom" "Phylum"  "Class"   "Order"   "Family"  "Genus"   "Species"
```

```
# list column meta data
names(colData(tse))
```

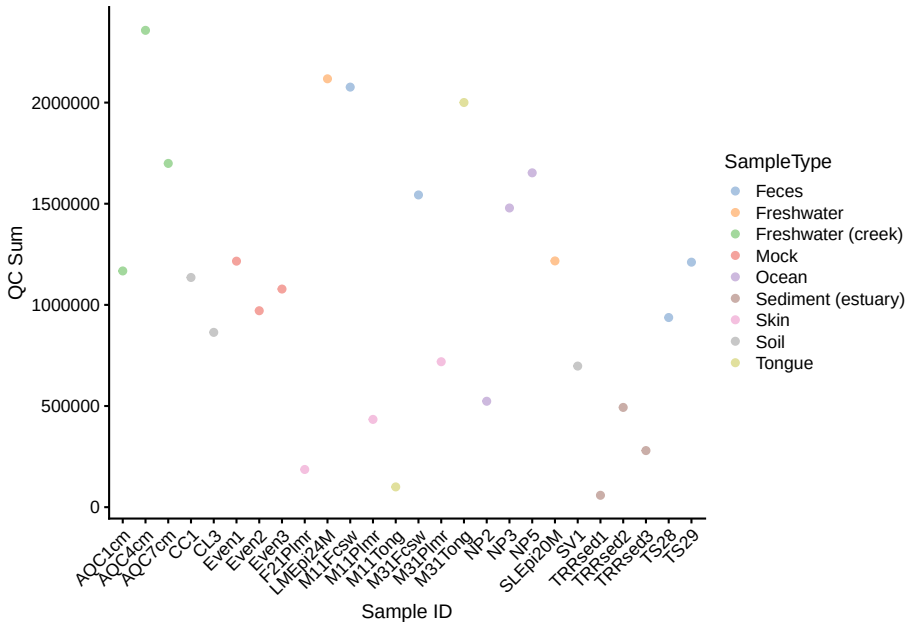
```
## [1] "X.SampleID"               "Primer"
## [3] "Final_Barcode"            "Barcode_truncated_plus_T"
## [5] "Barcode_full_length"      "SampleType"
## [7] "Description"
```

Such meta data can be directly plotted with the functions `plotRowData` and `plotColData`.

```
# obtain QC data
tse <- addPerCellQC(tse)
tse <- addPerFeatureQC(tse)
# plot QC Mean against Species
plotRowData(tse, "mean", "Species") +
  theme(axis.text.x = element_blank()) +
  labs(x = "Species", y = "QC Mean")
```

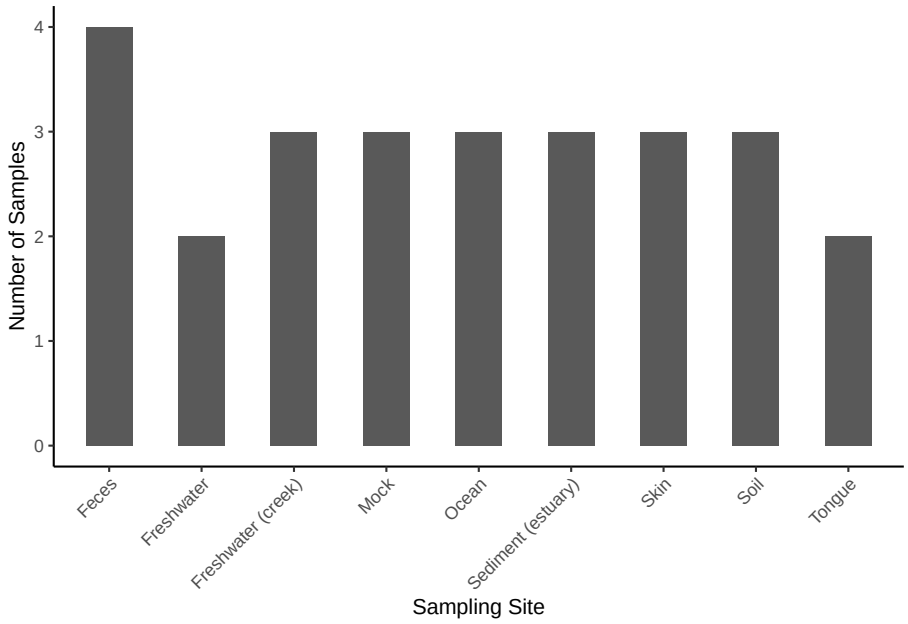



```
# plot QC Sum against Sample ID, colour-labeled by Sample Type
plotColData(tse, "sum", "X.SampleID", colour_by = "SampleType") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "Sample ID", y = "QC Sum")
```



Alternatively, they can be converted to a `data.frame` object and passed to `ggplot`.

```
# store colData into a data frame
coldata <- as.data.frame(colData(tse))
# plot Number of Samples against Sampling Site
ggplot(coldata, aes(x = SampleType)) +
  geom_bar(width = 0.5) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "Sampling Site",
       y = "Number of Samples")
```



Further methods of application can be found in the chapters 5.3 and 7.1.1 and in a few external tutorials with open data. Additionally, `rowData` and `colData` allow manipulation and subsetting of large data sets into smaller units, as explained in chapter 4.

14.1.2 Viewing abundance and prevalence patterns

Prior-to-analysis exploration may involve questions such as how microorganisms are distributed across samples (abundance) and what microorganisms are present in most of the samples (prevalence). The information on abundance and prevalence can be summarized into a **jitter** or **density plot** and a **tree**, respectively, with the *miaViz* package.

Specifically, the functions `plotAbundance`, `plotAbundanceDensity` and `plotRowTree` are used, and examples on their usage are discussed throughout chapter 5.

14.2 Diversity estimation

Alpha diversity is commonly measured as one of the diversity indices explained in chapter 7. Because the focus lies on each sample separately, one-dimensional plots, such as **scatter**, **violin** and **box plots**, are suitable.

Beta diversity is generally evaluated as one of the dissimilarity indices reported in chapter 8. Unlike alpha diversity, samples are compared collectively to estimate the heterogeneity across them, therefore multidimensional plots, such as **Shepard** and **ordination plots** are suitable.

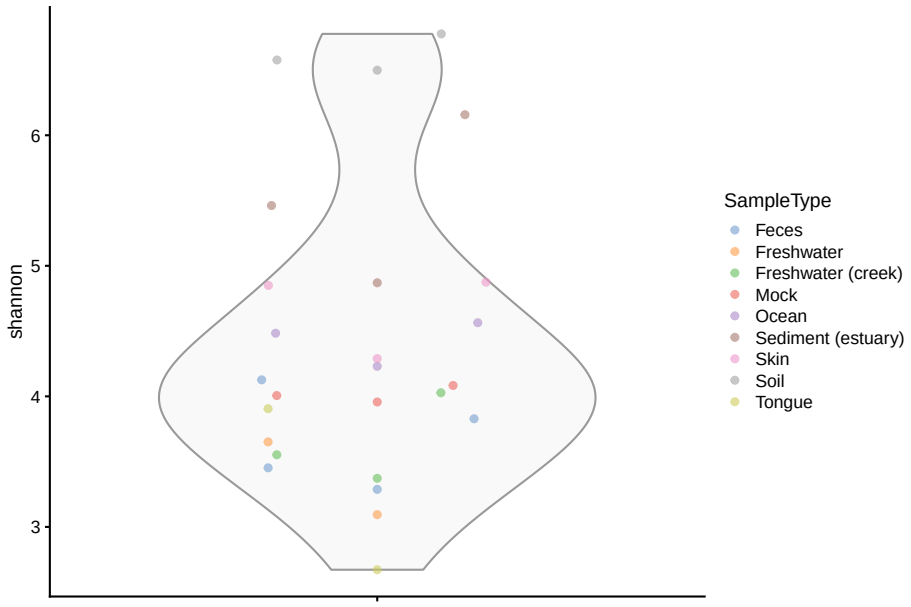
	alpha diversity	beta diversity
used metrics	diversity indices	dissimilarity indices
metric dimensionality	one-dimensional	multidimensional
suitable visualization	scatter, violin, box plots	Shepard, ordination plots

In conclusion, visualization techniques for alpha and beta diversity significantly differ from one another.

14.2.1 Alpha diversity with scatter, violin and box plots

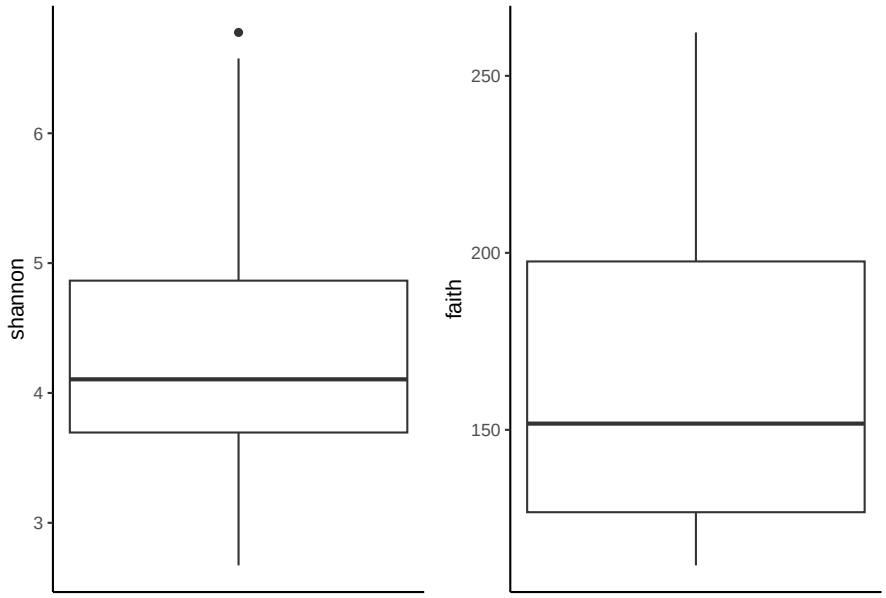
The basic method to visualize the diversity values assigned to the different samples in a `TSE` object includes the following, where each data point represents one sample:

```
# estimate shannon diversity index
tse <- mia::estimateDiversity(tse,
                              assay_name = "counts",
                              index = "shannon",
                              name = "shannon")
# plot shannon diversity index, colour-labeled by Sample Type
plotColData(tse, "shannon", colour_by = "SampleType")
```



The several indices available for the evaluation of alpha diversity often return slightly divergent results, which can be visually compared with a multiple violin or box plot. For this purpose, `plotColData` (for violin plots) or `ggplot` (for box plots) are recursively applied to a number of diversity indices with the function `lapply` and the multi-panel plotting functionality of the *patchwork* package is then exploited.

```
# estimate faith diversity index
tse <- mia::estimateFaith(tse,
                          assay_name = "counts")
# store colData into a data frame
coldata <- as.data.frame(colData(tse))
# generate plots for shannon and faith indices
# and store them into a list
plots <- lapply(c("shannon", "faith"),
                function(i) ggplot(coldata, aes_string(y = i)) +
                  geom_boxplot() +
                  theme(axis.text.x = element_blank(),
                        axis.ticks.x = element_blank()))
# combine plots with patchwork
plots[[1]] + plots[[2]]
```



The analogous output in the form of a violin plot is obtained in chapter 7.1.3. In addition, box plots that group samples according to certain information, such as origin, sex, age and health condition, can be labeled with p-values for significant differences with the package *ggsignif* package, as shown in chapter 7.1.2.

14.2.2 Beta diversity with Shepard and coordination plots

The *scater* package offers the general function `plotReducedDim`. In its basic form, it takes a `TSE` object and the results on sample similarity stored in the same object, which can be evaluated with the following coordination methods:

- `runMDS`
- `runNMDS`
- `runPCA`
- `runTSNE`
- `runUMAP`

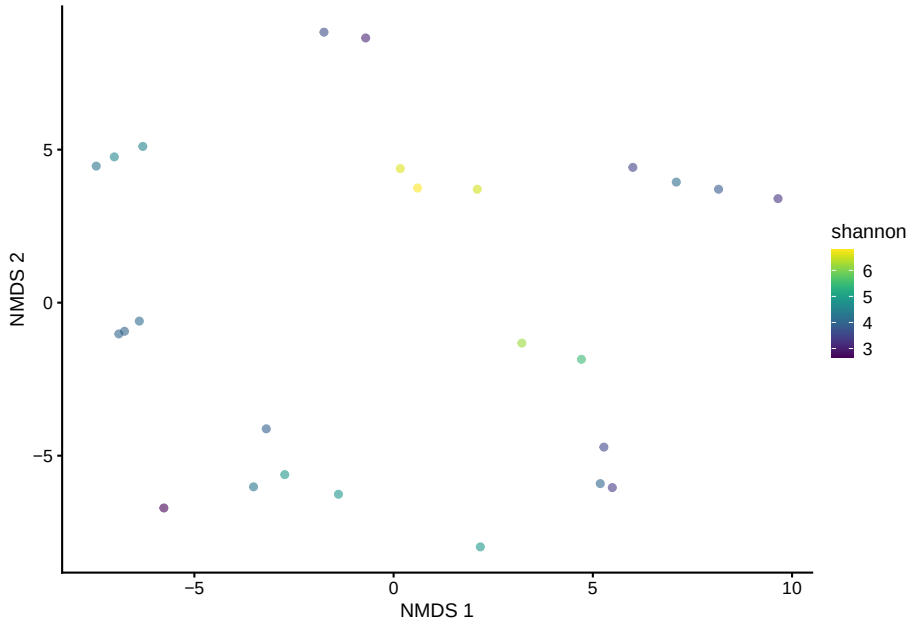
Since these clustering techniques allow for multiple coordinates or components, **coordination plots** can also span multiple dimensions, which is explained in chapter 17.

```
# perform NMDS coordination method
tse <- runNMDS(tse,
```

```
                FUN = vegan::vegdist,
                name = "NMDS")
```

```
## initial  value 47.733208
## iter   5 value 33.853364
## iter  10 value 32.891200
## final  value 32.823570
## converged
```

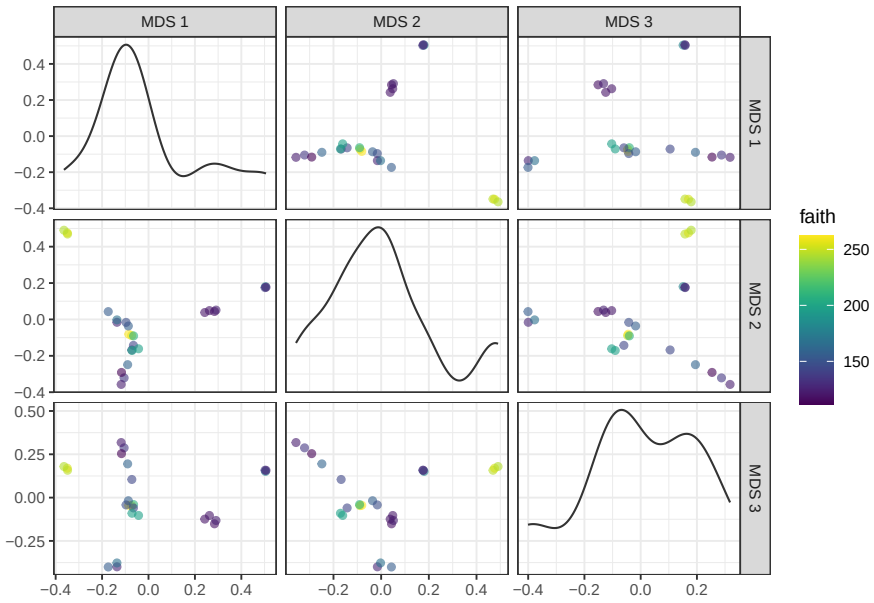
```
# plot results of a 2-component NMDS on tse,
# coloured-scaled by shannon diversity index
plotReducedDim(tse, "NMDS", colour_by = "shannon")
```



Multiple combinations of coordinates or dimensions can also be integrated into a multi-panel arrangement.

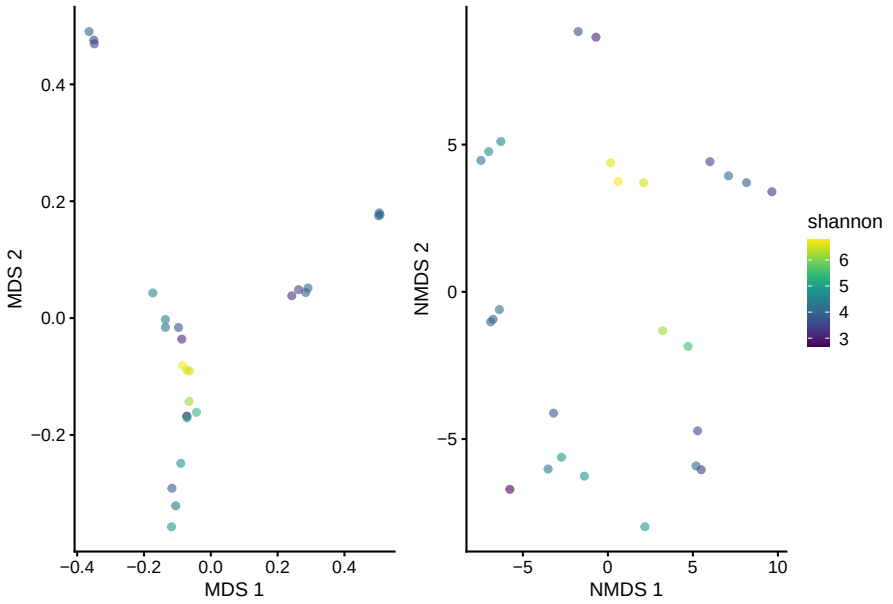
```
# perform MDS coordination method
tse <- runMDS(tse,
              FUN = vegan::vegdist,
              method = "bray",
              name = "MDS",
              exprs_values = "counts",
              ncomponents = 3)
```

```
# plot results of a 3-component MDS on tse,
# coloured-scaled by faith diversity index
plotReducedDim(tse, "MDS", ncomponents = c(1:3), colour_by =
 ↪  "faith")
```



Similarly to iterating `plotColData` over indices of alpha diversity, `lapply` can be used in combination with *patchwork* to recursively apply `plotReducedDim` and visually compare results among various coordination methods.

```
# generate plots for MDS and NMDS methods
# and store them into a list
plots <- lapply(c("MDS", "NMDS"),
                plotReducedDim,
                object = tse,
                colour_by = "shannon")
# combine plots with patchwork
plots[[1]] + plots[[2]] +
  plot_layout(guides = "collect")
```



For similar examples, readers are referred to chapter 8. Further material on the graphic capabilities of *patchwork* is available in its official package tutorial.

14.3 Statistical analysis

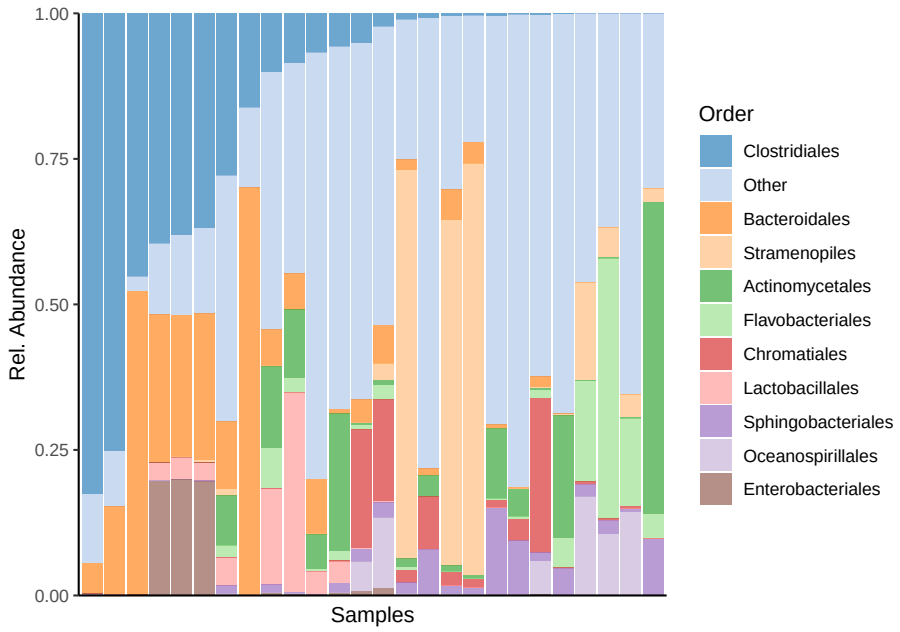
14.3.1 Heatmaps

As described in chapter 9.1, bar plots and heatmaps can offer a useful insight into the composition of a community. Simple methods involve the functions `plotAbundance` and `geom_tile` in combination with `scale_fill_gradientn` from the packages *miaViz* and *ggplot2*, respectively.

For instance, below the composition of multiple samples (x axis) is reported in terms of relative abundances (y axis) for the top 10 taxa at the Order rank. Bar plots and heatmaps with analogous information at the Phylum level are available in the aforementioned chapter.

```
# agglomerate tse by Order
tse_order <- agglomerateByRank(tse,
                               rank = "Order",
                               onRankOnly = TRUE)
# transform counts into relative abundance
tse_order <- transformCounts(tse_order,
                             assay_name = "counts",
```

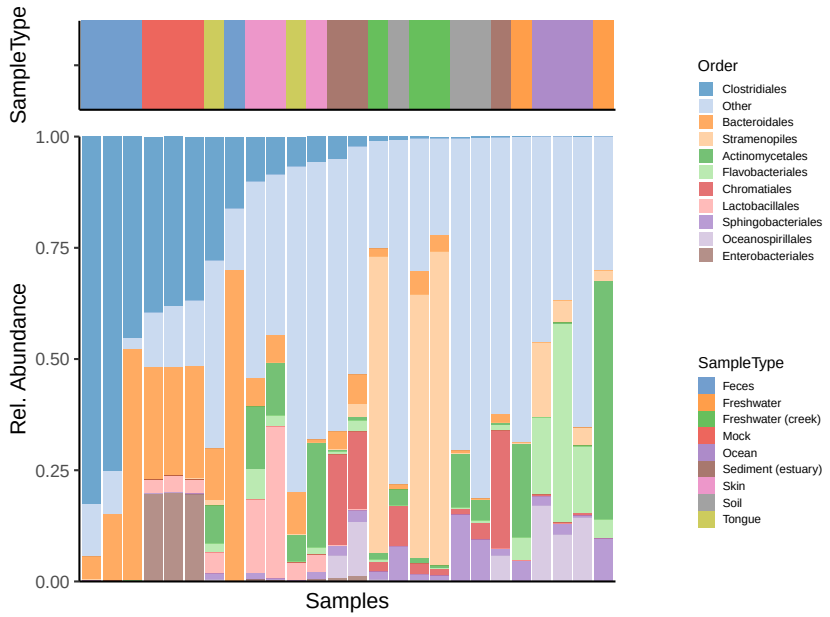
```
                                method = "relabundance")
# get top orders
top_taxa <- getTopTaxa(tse_order,
                       top = 10,
                       assay_name = "relabundance")
# leave only names for top 10 orders and label the rest with
 ↪  "Other"
order_renamed <- lapply(rowData(tse_order)$Order,
                    function(x){if (x %in% top_taxa) {x} else
                     ↪  {"Other"}})
rowData(tse_order)$Order <- as.character(order_renamed)
# plot composition as a bar plot
plotAbundance(tse_order,
              assay_name = "relabundance",
              rank = "Order",
              order_rank_by = "abund",
              order_sample_by = "Clostridiales")
```



To add a sample annotation, you can combine plots that you get from the output of *plotAbundance*.

```
# Create plots
plots <- plotAbundance(tse_order,
            assay_name = "relabundance",
```

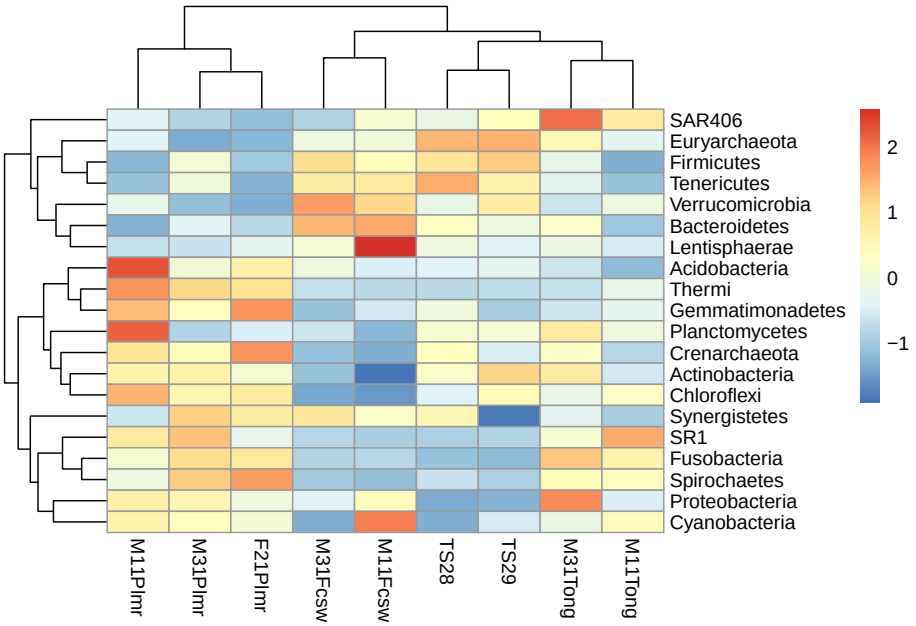
```
        rank = "Order",
            order_rank_by = "abund",
        order_sample_by = "Clostridiales",
            features = "SampleType")

# Modify the legend of the first plot to be smaller
plots[[1]] <- plots[[1]] +
    theme(legend.key.size = unit(0.3, 'cm'),
          legend.text = element_text(size = 6),
          legend.title = element_text(size = 8))

# Modify the legend of the second plot to be smaller
plots[[2]] <- plots[[2]] +
    theme(legend.key.height = unit(0.3, 'cm'),
          legend.key.width = unit(0.3, 'cm'),
          legend.text = element_text(size = 6),
          legend.title = element_text(size = 8),
          legend.direction = "vertical")

# Load required packages
if( !require("ggpubr") ){
    install.packages("ggpubr")
    library("ggpubr")
}
# Load required packages
if( !require("patchwork") ){
    install.packages("patchwork")
    library("patchwork")
}

# Combine legends
legend <- wrap_plots(as_ggplot(get_legend(plots[[1]])),
 ↪  as_ggplot(get_legend(plots[[2]])), ncol = 1)

# Remove legends from the plots
plots[[1]] <- plots[[1]] + theme(legend.position = "none")
plots[[2]] <- plots[[2]] + theme(legend.position = "none",
 ↪  axis.title.x=element_blank())

# Combine plots
plot <- wrap_plots(plots[[2]], plots[[1]], ncol = 1, heights =
 ↪  c(2, 10))
# Combine the plot with the legend
wrap_plots(plot, legend, nrow = 1, widths = c(2, 1))
```



For more sophisticated visualizations than those produced with `plotAbundance` and *ggplot2*, the packages *pheatmap* and *sechm* provide methods to include feature and sample clusters in a heatmap, along with further functionality.

```
# Agglomerate tse by phylum
tse_phylum <- agglomerateByRank(tse,
                                rank = "Phylum",
                                onRankOnly = TRUE)

# Add clr-transformation on samples
assay(tse_phylum, "pseudo") <- assay(tse_phylum, "counts") + 1
tse_phylum <- transformCounts(tse_phylum, assay_name = "pseudo",
                              method = "relabundance")
tse_phylum <- transformCounts(tse_phylum, method = "clr",
 ↪  assay_name = "relabundance")

# Add z-transformation on features (taxa)
tse_phylum <- transformCounts(tse_phylum, assay_name = "clr",
                              MARGIN = "features",
                              method = "z", name = "clr_z")

# Take subset: only samples from feces, skin, or tongue
tse_phylum_subset <- tse_phylum[ , tse_phylum$SampleType %in%
 ↪  c("Feces", "Skin", "Tongue") ]
```

```
# Add clr-transformation
assay(tse_phylum_subset, "pseudo") <- assay(tse_phylum_subset,
 ↪  "counts") + 1
tse_phylum_subset <- transformCounts(tse_phylum_subset, method =
 ↪  "relabundance",
                                     assay_name = "pseudo")
tse_phylum_subset <- transformCounts(tse_phylum_subset, method =
 ↪  "clr",
                                     assay_name = "relabundance")
# Does z-transformation
tse_phylum_subset <- transformCounts(tse_phylum_subset,
 ↪  assay_name = "clr",
                                     MARGIN = "features",
                                     method = "z", name =
                                      ↪  "clr_z")

# Get n most abundant taxa, and subsets the data by them
top_taxa <- getTopTaxa(tse_phylum_subset, top = 20)
tse_phylum_subset <- tse_phylum_subset[top_taxa, ]

# Gets the assay table
mat <- assay(tse_phylum_subset, "clr_z")

# Creates the heatmap
pheatmap(mat)
```



We can cluster both samples and features hierarchically and add them to the x and y axes of the heatmap, respectively.

```
# Hierarchical clustering
taxa_hclust <- hclust(dist(mat), method = "complete")

# Creates a phylogenetic tree
taxa_tree <- as.phylo(taxa_hclust)

# Plot taxa tree
taxa_tree <- ggtree(taxa_tree) +
  theme(plot.margin=margin(0,0,0,0)) # removes margins

# Get order of taxa in plot
taxa_ordered <- get_taxa_name(taxa_tree)

# to view the tree, run
# taxa_tree
```

Based on phylo tree, we decide to create three clusters.

```
# Creates clusters
taxa_clusters <- cutree(tree = taxa_hclust, k = 3)
```

```
# Converts into data frame
taxa_clusters <- data.frame(clusters = taxa_clusters)
taxa_clusters$clusters <- factor(taxa_clusters$clusters)

# Order data so that it's same as in phylo tree
taxa_clusters <- taxa_clusters[taxa_ordered, , drop = FALSE]

# Prints taxa and their clusters
taxa_clusters
```

```
##                   clusters
## Chloroflexi              3
## Actinobacteria           3
## Crenarchaeota            3
## Planctomycetes           3
## Gemmatimonadetes         3
## Thermi                   3
## Acidobacteria            3
## Spirochaetes             2
## Fusobacteria             2
## SR1                      2
## Cyanobacteria            2
## Proteobacteria           2
## Synergistetes            2
## Lentisphaerae            1
## Bacteroidetes            1
## Verrucomicrobia          1
## Tenericutes              1
## Firmicutes               1
## Euryarchaeota            1
## SAR406                   1
```

The information on the clusters is then added to the feature meta data.

```
# Adds information to rowData
rowData(tse_phylum_subset)$clusters <-
  taxa_clusters[order(match(rownames(taxa_clusters),
  rownames(tse_phylum_subset))), ]

# Prints taxa and their clusters
rowData(tse_phylum_subset)$clusters
```

```
##  [1] 1 1 2 3 2 2 1 1 1 1 3 2 3 3 3 2 2 3 3 1
## Levels: 1 2 3
```

Similarly, samples are hierarchically grouped into clusters, the most suitable number of clusters for the plot is selected and the new information is stored into the sample meta data.

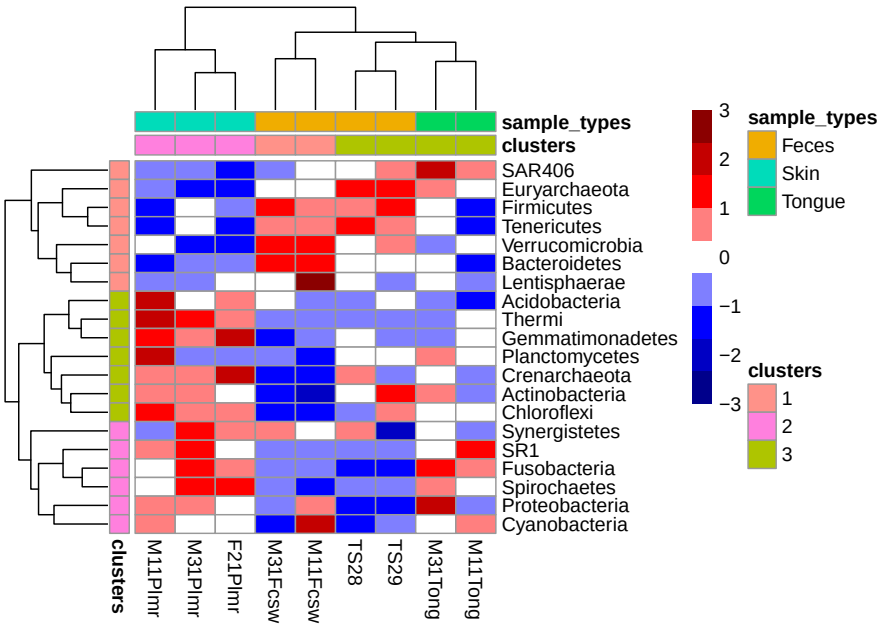
```
# Hierarchical clustering
sample_hclust <- hclust(dist(t(mat)), method = "complete")

# Creates a phylogenetic tree
sample_tree <- as.phylo(sample_hclust)

# Plot sample tree
sample_tree <- ggtree(sample_tree) + layout_dendrogram() +
  theme(plot.margin=margin(0,0,0,0)) # removes margins

# Get order of samples in plot
samples_ordered <- rev(get_taxa_name(sample_tree))

# to view the tree, run
# sample_tree

# Creates clusters
sample_clusters <- factor(cutree(tree = sample_hclust, k = 3))

# Converts into data frame
sample_data <- data.frame(clusters = sample_clusters)

# Order data so that it's same as in phylo tree
sample_data <- sample_data[samples_ordered, , drop = FALSE]

# Order data based on
tse_phylum_subset <- tse_phylum_subset[ , rownames(sample_data)]

# Add sample type data
sample_data$sample_types <-
 ↪  unfactor(colData(tse_phylum_subset)$SampleType)

sample_data
```

```
##         clusters sample_types
## M11Plmr        2         Skin
## M31Plmr        2         Skin
## F21Plmr        2         Skin
## M31Fcsw        1        Feces
## M11Fcsw        1        Feces
## TS28           3        Feces
```

```
## TS29            3         Feces
## M31Tong         3        Tongue
## M11Tong         3        Tongue
```

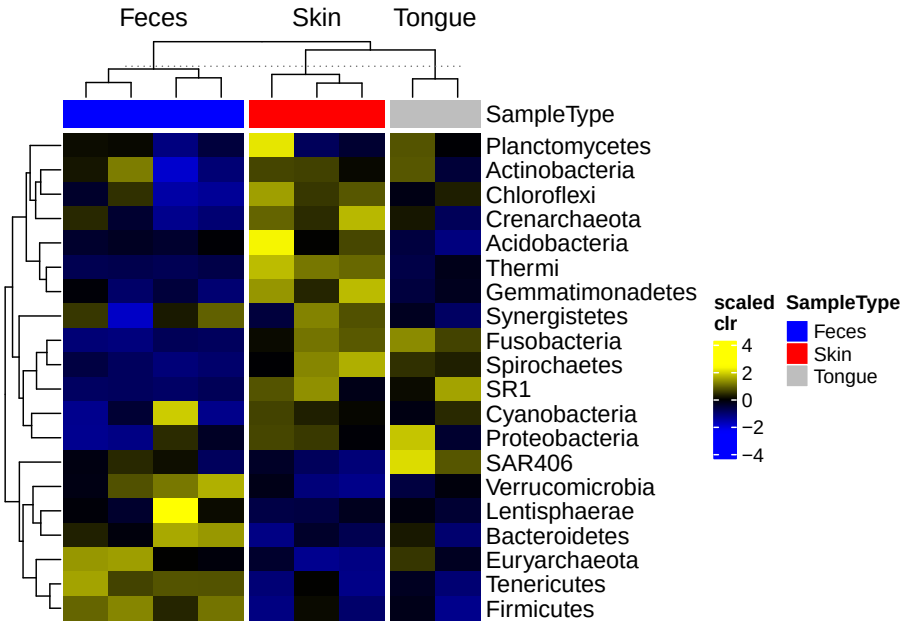
Now we can create heatmap with additional annotations.

```
# Determines the scaling of colorss
# Scale colors
breaks <- seq(-ceiling(max(abs(mat))), ceiling(max(abs(mat))),
              length.out = ifelse( max(abs(mat))>5,
               ↪  2*ceiling(max(abs(mat))), 10 ) )
colors <- colorRampPalette(c("darkblue", "blue", "white", "red",
 ↪  "darkred"))(length(breaks)-1)

pheatmap(mat, annotation_row = taxa_clusters,
         annotation_col = sample_data,
         breaks = breaks,
         color = colors)
```



The package *sechm* allows for further visual capabilities and flexibility. In this case, the clustering step is automatically performed by the plotting function and does not need to be executed in advance.

```
# Stores annotation colros to metadata
metadata(tse_phylum_subset)$anno_colors$SampleType <- c(Feces =
 ↪  "blue",
                                                        Skin =
                                                         ↪  "red",
                                                         ↪
                                                        Tongue =
                                                         ↪  "gray")

# Create a plot
sechm(tse_phylum_subset,
      features = rownames(tse_phylum_subset),
      assayName = "clr",
      do.scale = TRUE,
      top_annotation = c("SampleType"),
      gaps_at = "SampleType",
      cluster_cols = TRUE, cluster_rows = TRUE)
```



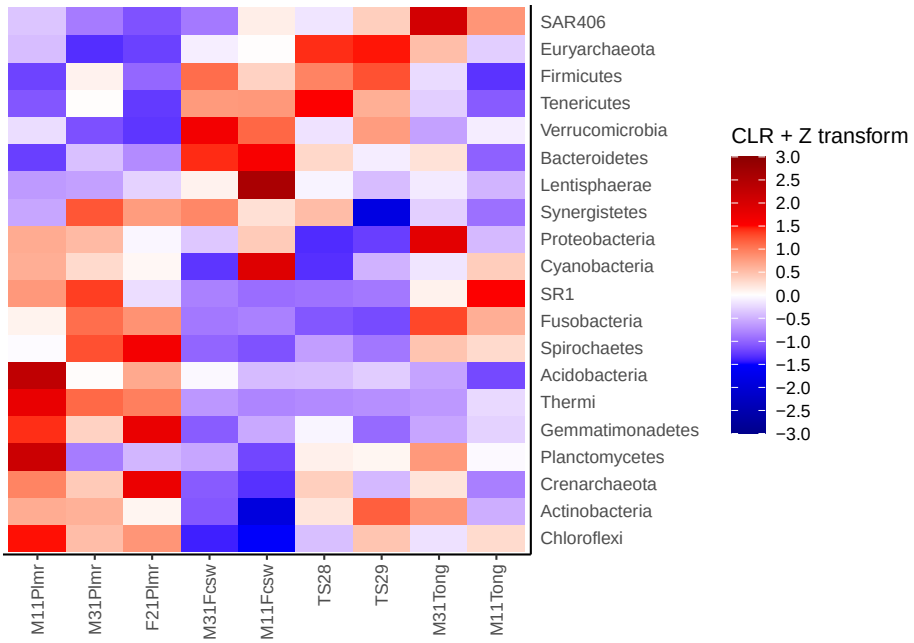
It is also possible to create an analogous heatmap by just using the *ggplot2* package. However, a relatively long code is required to generate an identical output.

```
# Add feature names to column as a factor
taxa_clusters$Feature <- rownames(taxa_clusters)
```

```
taxa_clusters$Feature <- factor(taxa_clusters$Feature, levels =
 ↪  taxa_clusters$Feature)

# Create annotation plot
row_annotation <- ggplot(taxa_clusters) +
  geom_tile(aes(x = NA, y = Feature, fill = clusters)) +
  coord_equal(ratio = 1) +
  theme(
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        axis.title.y=element_blank(),
        axis.title.x = element_text(angle = 90, vjust = 0.5,
         ↪  hjust=1),
        plot.margin=margin(0,0,0,0),
        ) +
      labs(fill = "Clusters", x = "Clusters")

# to view the notation, run
# row_annotation

# Add sample names to one of the columns
sample_data$sample <- factor(rownames(sample_data), levels =
 ↪  rownames(sample_data))

# Create annotation plot
sample_types_annotation <- ggplot(sample_data) +
  scale_y_discrete(position = "right", expand = c(0,0)) +
  geom_tile(aes(y = NA, x = sample, fill = sample_types)) +
  coord_equal(ratio = 1) +
  theme(
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.title.x=element_blank(),
        axis.ticks.x=element_blank(),
        plot.margin=margin(0,0,0,0),
        axis.title.y.right = element_text(angle=0, vjust = 0.5)
        ) +
      labs(fill = "Sample types", y = "Sample types")
# to view the notation, run
# sample_types_annotation

# Create annotation plot
sample_clusters_annotation <- ggplot(sample_data) +
  scale_y_discrete(position = "right", expand = c(0,0)) +
```

```
  geom_tile(aes(y = NA, x = sample, fill = clusters)) +
  coord_equal(ratio = 1) +
  theme(
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.title.x=element_blank(),
        axis.ticks.x=element_blank(),
        plot.margin=margin(0,0,0,0),
        axis.title.y.right = element_text(angle=0, vjust = 0.5)
        ) +
      labs(fill = "Clusters", y = "Clusters")
# to view the notation, run
# sample_clusters_annotation

# Order data based on clusters and sample types
mat <- mat[unfactor(taxa_clusters$Feature),
 ↪  unfactor(sample_data$sample)]

# ggplot requires data in melted format
melted_mat <- melt(mat)
colnames(melted_mat) <- c("Taxa", "Sample", "clr_z")

# Determines the scaling of colorss
maxval <- round(max(abs(melted_mat$clr_z)))
limits <- c(-maxval, maxval)
breaks <- seq(from = min(limits), to = max(limits), by = 0.5)
colours <- c("darkblue", "blue", "white", "red", "darkred")

heatmap <- ggplot(melted_mat) +
  geom_tile(aes(x = Sample, y = Taxa, fill = clr_z)) +
  theme(
    axis.title.y=element_blank(),
    axis.title.x=element_blank(),
    axis.ticks.y=element_blank(),
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),

    plot.margin=margin(0,0,0,0), # removes margins
    legend.key.height= unit(1, 'cm')
    ) +
  scale_fill_gradientn(name = "CLR + Z transform",
                       breaks = breaks,
                       limits = limits,
                       colours = colours) +
  scale_y_discrete(position = "right")
```

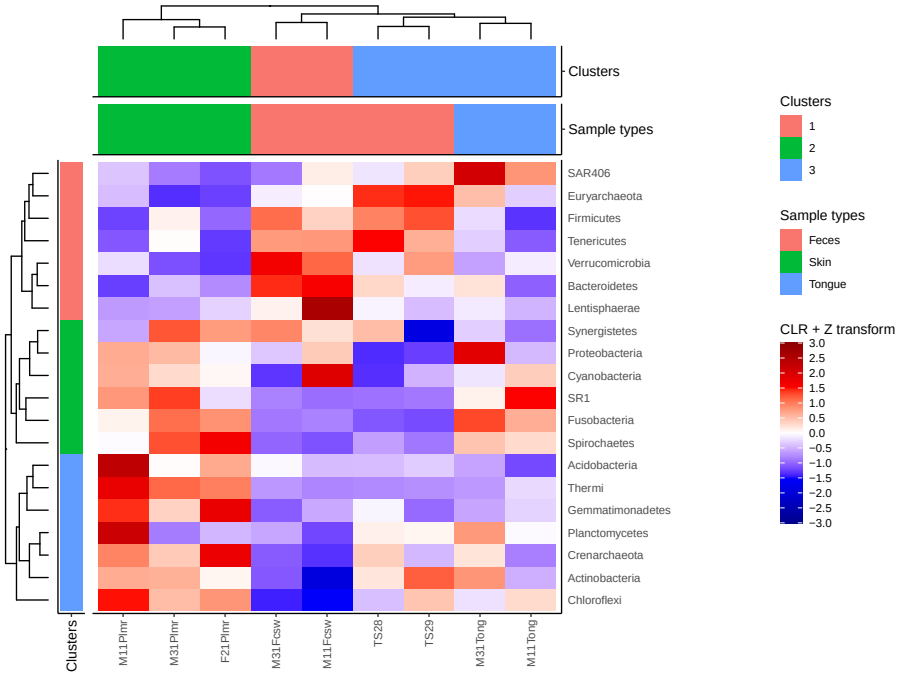
heatmap




```
# Create layout
design <- c(
  area(3, 1, 4, 1),
  area(1, 2, 1, 3),
  area(2, 2, 2, 3),
  area(3, 2, 4, 3)
)
# to view the design, run
# plot(design)

# Combine plots
plot <- row_annotation + sample_clusters_annotation +
                         sample_types_annotation +
              heatmap  +
    plot_layout(design = design, guides = "collect",
                # Specify layout, collect legends

                # Adjust widths and heights to align plots.
                # When annotation plot is larger, it might not
                 ↪  fit into
        # its column/row.
                # Then you need to make column/row larger.
```

```
                # Relative widths and heights of each column and
                 ↪  row:
                # Currently, the width of the first column is 15
                 ↪  % and the height of
                # first two rows are 30 % the size of others

                # To get this work most of the times, you can
                 ↪  adjust all sizes to be 1, i.e. equal,
                # but then the gaps between plots are larger.
                widths = c(0.15, 1, 1),
                heights = c(0.3, 0.3, 1, 1))

# plot
```

```
# Create layout
design <- c(
  area(4, 1, 5, 1),
  area(4, 2, 5, 2),
  area(1, 3, 1, 4),
  area(2, 3, 2, 4),
  area(3, 3, 3, 4),
  area(4, 3, 5, 4)
)

# to view the design, run
# plot(design)

# Combine plots
plot <- taxa_tree +
  row_annotation +
  sample_tree +
  sample_clusters_annotation +
  sample_types_annotation +
  heatmap +
    plot_layout(design = design, guides = "collect", # Specify
     ↪  layout, collect legends
                widths = c(0.2, 0.15, 1, 1, 1),
                heights = c(0.1, 0.15, 0.15, 0.25, 1, 1))

plot
```



Heatmaps find several other applications in biclustering and multi-assay analyses, that are discussed in chapters 10 and 13, where the packages *cobiclust* and *MOFA2* are of interest.

Session Info

View session info

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
```

```
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] ggpubr_0.6.0                    ggtree_3.4.4
 [3] ape_5.7-1                       pheatmap_1.0.12
 [5] reshape2_1.4.4                  sechm_1.4.1
 [7] miaViz_1.7.5                    ggraph_2.1.0
 [9] patchwork_1.1.2                 scater_1.26.1
[11] scuttle_1.8.4                   mia_1.7.11
[13] MultiAssayExperiment_1.24.0     TreeSummarizedExperiment_2.1.4
[15] Biostrings_2.66.0               XVector_0.38.0
[17] SingleCellExperiment_1.20.1     SummarizedExperiment_1.28.0
[19] Biobase_2.58.0                  GenomicRanges_1.50.2
[21] GenomeInfoDb_1.34.9             IRanges_2.32.0
[23] S4Vectors_0.36.2                BiocGenerics_0.44.0
[25] MatrixGenerics_1.10.0           matrixStats_0.63.0-9003
[27] ggplot2_3.4.1                   BiocStyle_2.24.0
[29] rebook_1.6.0

loaded via a namespace (and not attached):
  [1] backports_1.4.1             circlize_0.4.15
  [3] plyr_1.8.8                  igraph_1.4.1
  [5] lazyeval_0.2.2              splines_4.2.1
  [7] BiocParallel_1.32.6         digest_0.6.31
  [9] ca_0.71.1                   foreach_1.5.2
 [11] yulab.utils_0.0.6           htmltools_0.5.5
 [13] viridis_0.6.2               fansi_1.0.4
 [15] magrittr_2.0.3              memoise_2.0.1
 [17] ScaledMatrix_1.6.0          cluster_2.1.4
 [19] doParallel_1.0.17           DECIPHER_2.26.0
 [21] ComplexHeatmap_2.12.1       graphlayouts_0.8.4
 [23] colorspace_2.1-0            blob_1.2.4
 [25] ggrepel_0.9.3               xfun_0.38
 [27] dplyr_1.1.1                 crayon_1.5.2
 [29] RCurl_1.98-1.12             jsonlite_1.8.4
 [31] graph_1.74.0                iterators_1.0.14
 [33] glue_1.6.2                  polyclip_1.10-4
 [35] registry_0.5-1              gtable_0.3.3
 [37] zlibbioc_1.44.0             V8_4.2.2
 [39] GetoptLong_1.0.5            DelayedArray_0.24.0
```

```
 [41] car_3.1-1                  BiocSingular_1.14.0
 [43] shape_1.4.6                abind_1.4-5
 [45] scales_1.2.1               DBI_1.1.3
 [47] rstatix_0.7.2              randomcoloR_1.1.0.1
 [49] Rcpp_1.0.10                viridisLite_0.4.1
 [51] clue_0.3-64                decontam_1.18.0
 [53] gridGraphics_0.5-1         tidytree_0.4.2
 [55] bit_4.0.5                  rsvd_1.0.5
 [57] RColorBrewer_1.1-3         dir.expiry_1.4.0
 [59] pkgconfig_2.0.3            XML_3.99-0.14
 [61] farver_2.1.1               CodeDepends_0.6.5
 [63] utf8_1.2.3                 labeling_0.4.2
 [65] ggplotify_0.1.0            tidyselect_1.2.0
 [67] rlang_1.1.0                munsell_0.5.0
 [69] tools_4.2.1                cachem_1.0.7
 [71] cli_3.6.1                  DirichletMultinomial_1.40.0
 [73] generics_0.1.3             RSQLite_2.3.0
 [75] broom_1.0.4                evaluate_0.20
 [77] stringr_1.5.0              fastmap_1.1.1
 [79] yaml_2.3.7                 knitr_1.42
 [81] bit64_4.0.5                tidygraph_1.2.3
 [83] purrr_1.0.1                nlme_3.1-162
 [85] sparseMatrixStats_1.10.0   aplot_0.1.10
 [87] compiler_4.2.1             curl_5.0.0
 [89] beeswarm_0.4.0             filelock_1.0.2
 [91] png_0.1-8                  ggsignif_0.6.4
 [93] treeio_1.22.0              tibble_3.2.1
 [95] tweenr_2.0.2               stringi_1.7.12
 [97] highr_0.10                 lattice_0.20-45
 [99] Matrix_1.5-3               vegan_2.6-4
[101] permute_0.9-7              vctrs_0.6.1
[103] pillar_1.9.0               lifecycle_1.0.3
[105] BiocManager_1.30.20        GlobalOptions_0.1.2
[107] BiocNeighbors_1.16.0       cowplot_1.1.1
[109] bitops_1.0-7               irlba_2.3.5.1
[111] seriation_1.4.2            R6_2.5.1
[113] TSP_1.2-3                  bookdown_0.33
[115] gridExtra_2.3              vipor_0.4.5
[117] codetools_0.2-19           MASS_7.3-58.3
[119] rjson_0.2.21               withr_2.5.0
[121] GenomeInfoDbData_1.2.9     mgcv_1.8-42
[123] parallel_4.2.1             grid_4.2.1
[125] ggfun_0.0.9                beachmat_2.14.0
[127] tidyr_1.3.0                rmarkdown_2.21
[129] DelayedMatrixStats_1.20.0  carData_3.0-5
[131] Cairo_1.6-0                Rtsne_0.16
```

```
[133] ggnewscale_0.4.8            ggforce_0.4.1
[135] ggbeeswarm_0.7.1
```

Part III

Appendix

Chapter 15

Training

The tutorial can support course teaching in R/Bioconductor.

15.1 Code of Conduct

We support the Bioconductor Code of Conduct. In short, the Bioconductor community values an open approach to science that promotes

```
* sharing of ideas, code, software and expertise
* collaboration
* diversity and inclusivity
* a kind and welcoming environment
* community contributions
```

For further details, we refer the reader to the full Code of Conduct.

15.2 Checklist

15.2.1 Setting up Your own computer

Setting up the system on your own computer is not required for the course but it can be useful for later use. The required software:

- R (the latest official release)
- RStudio; choose "Rstudio Desktop" to download the latest version. RStudio is optional. Check the Rstudio home page for more information.

- Install and load essential R packages (see Section 15.3)
- After a successful installation you can start with the case study examples in Section 19.

15.2.2 Support and resources

- Check additional reading tips and try out online material listed in Section 15.2.3.
- **You can run the workflows by simply copy-pasting the examples.** You can then test further examples from this tutorial, modifying and applying these techniques to your own data.
- Online support on installation and other matters, join us at Gitter

15.2.3 Study material

We encourage to familiarize with the material and test examples in advance.

- Orchestrating Microbiome Analysis with R/Bioconductor (OMA)
- Short online videos on R/Bioconductor microbiome data science tools.
- Lecture slides will be made available online
- Exercises
- Resources

15.3 Installing and loading the required R packages

This section shows how to install and load all required packages into the R session, if needed.

```
# List of packages that we need from cran and bioc
cran_pkg <- c("BiocManager", "bookdown", "dplyr", "ecodist",
 ↪  "ggplot2",
              "gridExtra", "kableExtra",  "knitr", "scales",
               ↪  "vegan", "matrixStats")
bioc_pkg <- c("yulab.utils","ggtree","ANCOMBC", "ape", "DESeq2",
 ↪  "DirichletMultinomial", "mia", "miaViz")
```

```
# Get those packages that are already installed
cran_pkg_already_installed <- cran_pkg[ cran_pkg %in%
 ↪  installed.packages() ]
bioc_pkg_already_installed <- bioc_pkg[ bioc_pkg %in%
 ↪  installed.packages() ]

# Get those packages that need to be installed
cran_pkg_to_be_installed <- setdiff(cran_pkg,
 ↪  cran_pkg_already_installed)
bioc_pkg_to_be_installed <- setdiff(bioc_pkg,
 ↪  bioc_pkg_already_installed)

# Reorders bioc packages, so that mia and miaViz are first
bioc_pkg <- c(bioc_pkg[ bioc_pkg %in% c("mia", "miaViz") ],
              bioc_pkg[ !bioc_pkg %in% c("mia", "miaViz") ] )

# Combine to one vector
packages <- c(bioc_pkg, cran_pkg)
packages_to_install <- c( bioc_pkg_to_be_installed,
 ↪  cran_pkg_to_be_installed )
```

```
# If there are packages that need to be installed, install them
if( length(packages_to_install) ) {
   BiocManager::install(packages_to_install)
}
```

Now all required packages are installed, so let's load them into the session. Some function names occur in multiple packages. That is why we have prioritized the packages mia and miaViz on the list. Packages that are loaded first have a higher priority.

```
# Loading all packages into session. Returns true if package was
 ↪  successfully loaded.
loaded <- sapply(packages, require, character.only = TRUE)
as.data.frame(loaded)
```

```
##                      loaded
## mia                    TRUE
## miaViz                 TRUE
## yulab.utils            TRUE
## ggtree                 TRUE
## ANCOMBC                TRUE
## ape                    TRUE
```

```
## DESeq2                TRUE
## DirichletMultinomial  TRUE
## BiocManager           TRUE
## bookdown              TRUE
## dplyr                 TRUE
## ecodist               TRUE
## ggplot2               TRUE
## gridExtra             TRUE
## kableExtra            TRUE
## knitr                 TRUE
## scales                TRUE
## vegan                 TRUE
## matrixStats           TRUE
```

Chapter 16

Resources

16.1 Data containers

16.1.1 Resources for TreeSummarizedExperiment

- SingleCellExperiment (Lun and Risso, 2020)
  - Online tutorial
  - Project page
- SummarizedExperiment (Morgan et al., 2020)
  - Online tutorial
  - Project page
- TreeSummarizedExperiment (Huang, 2020)
  - Online tutorial
  - Project page
  - Publication: (Huang et al., 2021)

16.1.2 Other relevant containers

- DataFrame which behaves similarly to `data.frame`, yet efficient and fast when used with large datasets.
- DNAString along with `DNAStringSet`,`RNAString` and `RNAStringSet` efficient storage and handling of long biological sequences are offered within the Biostrings package (Pagès et al., 2020).
- GenomicRanges ((Lawrence et al., 2013)) offers an efficient representation and manipulation of genomic annotations and alignments, see e.g. `GRanges` and `GRangesList` at An Introduction to the GenomicRangesPackage.

NGS Analysis Basics provides a walk-through of the above-mentioned features with detailed examples.

16.1.3 Alternative containers for microbiome data

The `phyloseq` package and class became the first widely used data container for microbiome data science in R. Many methods for taxonomic profiling data are readily available for this class. We provide here a short description how `phyloseq` and `*Experiment` classes relate to each other.

`assays` : This slot is similar to `otu_table` in `phyloseq`. In a `SummarizedExperiment` object multiple assays, raw counts, transformed counts can be stored. See also (Ramos et al., 2017) for storing data from multiple `experiments` such as RNASeq, Proteomics, etc. `rowData` : This slot is similar to `tax_table` in `phyloseq` to store taxonomic information. `colData` : This slot is similar to `sample_data` in `phyloseq` to store information related to samples. `rowTree` : This slot is similar to `phy_tree` in `phyloseq` to store phylogenetic tree.

In this book, you will come across terms like `FeatureIDs` and `SampleIDs`. `FeatureIDs` : These are basically OTU/ASV ids which are row names in `assays` and `rowData`. `SampleIDs` : As the name suggests, these are sample ids which are column names in `assays` and row names in `colData`.

`FeatureIDs` and `SampleIDs` are used but the technical terms `rownames` and `colnames` are encouraged to be used, since they relate to actual objects we work with.

16.1.4 Resources for phyloseq

The (Tree)SummarizedExperiment objects can be converted into the alternative phyloseq format, for which further methods are available.

- List of R tools for microbiome analysis
- phyloseq (McMurdie and Holmes, 2013)
- microbiome tutorial
- microbiomeutilities
- Bioconductor Workflow for Microbiome Data Analysis: from raw reads to community analyses (Callahan et al., 2016b).

16.2 R programming resources

If you are new to R, you could try swirl for a kickstart to R programming. Further support resources are available through the Bioconductor project (Huber et al., 2015).

- R programming basics: Base R
- Basics of R programming: Base R
- R cheat sheets
- R visualization with ggplot2
- R graphics cookbook

Rmarkdown

- Rmarkdown tips (Xie et al., 2020)

RStudio

- RStudio cheat sheet

16.2.1 Bioconductor Classes

S4 system

S4 class system has brought several useful features to the object-oriented programming paradigm within R, and it is constantly deployed in R/Bioconductor packages (Huber et al., 2015).

Online Document:

- Hervé Pagès, A quick overview of the S4 class system.
- Laurent Gatto, A practical tutorial on S4 programming
- How S4 Methods Work (Chambers, 2006)

Books:

- John M. Chambers. Software for Data Analysis: Programming with R. Springer, New York, 2008. ISBN-13 978-0387759357 (Chambers, 2008)
- I Robert Gentleman. R Programming for Bioinformatics. Chapman & Hall/CRC, New York, 2008. ISBN-13 978-1420063677 (Gentleman, 2008)

16.3 Reproducible reporting with Quarto

Reproducible reporting is the starting point for robust interactive data science. Perform the following tasks:

- If you are entirely new to literate programming and reproducible reporting, take this 10 minute tutorial to get introduced to the most important functions within Markdown.

- Create a Quarto template in RStudio, and render it into a document (markdown, PDF, docx or other format). In case you are new to Quarto, see this page.
- Examples are tips for closely related Rmarkdown are available in the online tutorial to reproducible reporting by Dr. C Titus Brown; see also Rmarkdown cheatsheet

Figure sources:

Original article - Huang R *et al.* (2021) TreeSummarizedExperiment: a S4 class for data with hierarchical structure. F1000Research 9:1246. (Huang et al., 2021)

Reference Sequence slot extension - Lahti L *et al.* (2020) Upgrading the R/Bioconductor ecosystem for microbiome research F1000Research 9:1464 (slides).

Chapter 17

Extra material

17.1 PERMANOVA comparison

Here we present two possible uses of the `adonis2` function which performs PERMANOVA. The optional argument `by` has an effect on the statistical outcome, so its two options are compared here.

```
# import necessary packages
if (!require(gtools)){
  install.packages("gtools")
}
if (!require(purrr)){
  install.packages("purrr")
}
library(vegan)
library(gtools)
library(purrr)
```

Let us load the *enterotype* TSE object and run PERMANOVA for different orders of three variables with two different approaches: `by = "margin"` or `by = "terms"`.

```
# load and prepare data
library(mia)
data("enterotype", package="mia")
enterotype <- transformCounts(enterotype, method =
 ↪  "relabundance")
# drop samples missing meta data
enterotype <- enterotype[ , !rowSums(is.na(colData(enterotype)[,
 ↪  c("Nationality", "Gender", "ClinicalStatus")]) > 0)]
```

```
# define variables and list all possible combinations
vars <- c("Nationality", "Gender", "ClinicalStatus")
var_perm <- permutations(n = 3, r = 3, vars)
formulas <- apply(var_perm, 1, function(row) purrr::reduce(row,
 ↪  function(x, y) paste(x, "+", y)))
# create empty data.frames for further storing p-values
terms_df <- data.frame("Formula" = formulas,
                       "ClinicalStatus" = rep(0, 6),
                       "Gender" = rep(0, 6),
                       "Nationality" = rep(0, 6))
margin_df <- data.frame("Formula" = formulas,
                        "ClinicalStatus" = rep(0, 6),
                        "Gender" = rep(0, 6),
                        "Nationality" = rep(0, 6))
```

```
for (row_idx in 1:nrow(var_perm)) {

  # generate temporary formula (i.e. "assay ~ ClinicalStatus +
   ↪  Nationality + Gender")
  tmp_formula <- purrr::reduce(var_perm[row_idx, ], function(x,
 ↪  y) paste(x, "+", y))
  tmp_formula <- as.formula(paste0('t(assay(enterotype,
 ↪  "relabundance")) ~ ',
                                   tmp_formula))

  # multiple variables, default: by = "terms"
  set.seed(75)
  with_terms <- adonis2(tmp_formula,
                by = "terms",
                data = colData(enterotype),
                permutations = 99)

  # multiple variables, by = "margin"
  set.seed(75)
  with_margin <- adonis2(tmp_formula,
                 by = "margin",
                 data = colData(enterotype),
                 permutations = 99)

  # extract p-values
  terms_p <- with_terms[["Pr(>F)"]]
  terms_p <- terms_p[!is.na(terms_p)]
  margin_p <- with_margin[["Pr(>F)"]]
  margin_p <- margin_p[!is.na(margin_p)]
```

```
  # store p-values into data.frames
  for (col_idx in 1:ncol(var_perm)) {

    terms_df[var_perm[row_idx, col_idx]][row_idx, ] <-
↪   terms_p[col_idx]
    margin_df[var_perm[row_idx, col_idx]][row_idx, ] <-
↪   margin_p[col_idx]

  }

}
```

The following table displays the p-values for the three variables ClinicalStatus, Gender and Nationality obtained by PERMANOVA with `adonis2`. Note that the p-values remain identical when `by = "margin"`, but change with the order of the variables in the formula when `by = "terms"` (default).

```
df <- terms_df %>%
  dplyr::inner_join(margin_df, by = "Formula", suffix = c("
↪   (terms)", " (margin)"))

knitr::kable(df)
```

Formula	ClinicalStatus (terms)	Gender (terms)	Nationality (terms)	Clinica
ClinicalStatus + Gender + Nationality	0.20	0.70	0.04	
ClinicalStatus + Nationality + Gender	0.20	0.29	0.05	
Gender + ClinicalStatus + Nationality	0.17	0.79	0.04	
Gender + Nationality + ClinicalStatus	0.53	0.79	0.03	
Nationality + ClinicalStatus + Gender	0.61	0.29	0.04	
Nationality + Gender + ClinicalStatus	0.53	0.39	0.04	

17.2 Bayesian Multinomial Logistic-Normal Models

Analysis using such model could be performed with the function `pibble` from the `fido` package, wihch is in form of a Multinomial Logistic-Normal Linear Regression model; see vignette of package.

The following presents such an exemplary analysis based on the data of Sprockett et al. (2020) available through `microbiomeDataSets` package.

```
if (!require(fido)){
  # installing the fido package
  devtools::install_github("jsilve24/fido")
}
```

Loading the libraries and importing data:

```
library(fido)
```

```
library(microbiomeDataSets)
tse <- SprockettTHData()
```

We pick three covariates ("Sex","Age_Years","Delivery_Mode") during this analysis as an example, and beforehand we check for missing data:

```
library(mia)
cov_names <- c("Sex","Age_Years","Delivery_Mode")
na_counts <- apply(is.na(colData(tse)[,cov_names]), 2, sum)
na_summary<-as.data.frame(na_counts,row.names=cov_names)
```

We drop missing values of the covariates:

```
tse <- tse[ , !is.na(colData(tse)$Delivery_Mode) ]
tse <- tse[ , !is.na(colData(tse)$Age_Years) ]
```

We agglomerate microbiome data to Phylum:

```
tse_phylum <- agglomerateByRank(tse, "Phylum")
```

We extract the counts assay and covariate data to build the model matrix:

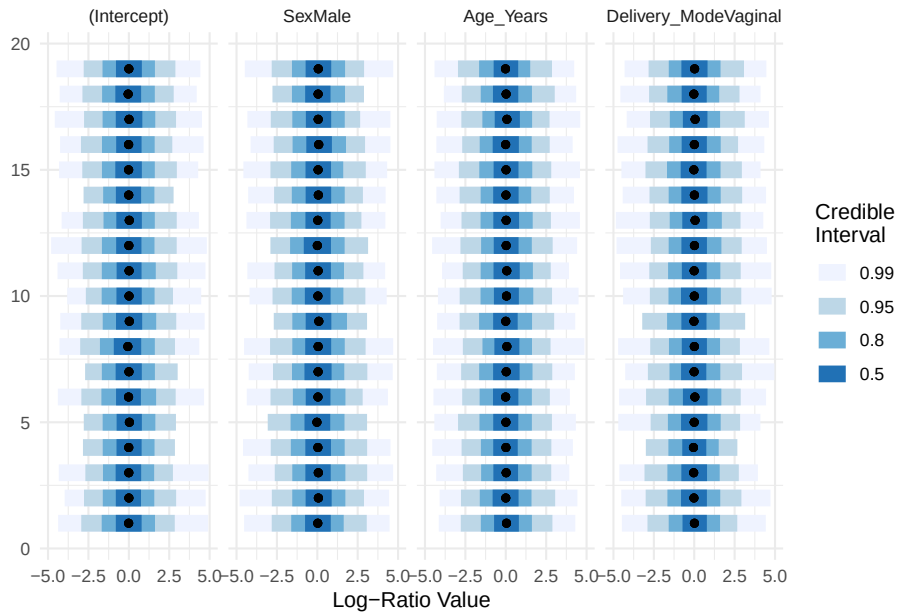
```
Y <- assays(tse_phylum)$counts
# design matrix
# taking 3 covariates
sample_data<-as.data.frame(colData(tse_phylum)[,cov_names])
X <-
 ↪  t(model.matrix(~Sex+Age_Years+Delivery_Mode,data=sample_data))
```

Building the parameters for the `pibble` call to build the model; see more at vignette:

```
n_taxa<-nrow(Y)
upsilon <- n_taxa+3
Omega <- diag(n_taxa)
G <- cbind(diag(n_taxa-1), -1)
Xi <- (upsilon-n_taxa)*G%*%Omega%*%t(G)
Theta <- matrix(0, n_taxa-1, nrow(X))
Gamma <- diag(nrow(X))
```

Automatically initializing the priors and visualizing their distributions:

```
priors <- pibble(NULL, X, upsilon, Theta, Gamma, Xi)
names_covariates(priors) <- rownames(X)
plot(priors, pars="Lambda") + ggplot2::xlim(c(-5, 5))
```



Estimating the posterior by including our response data `Y`. Note: Some computational failures could occur (see discussion) the arguments `multDirichletBoot` `calcGradHess` could be passed in such case.

```
priors$Y <- Y
posterior <- refit(priors, optim_method="adam",
 ↪  multDirichletBoot=0.5) #calcGradHess=FALSE
```

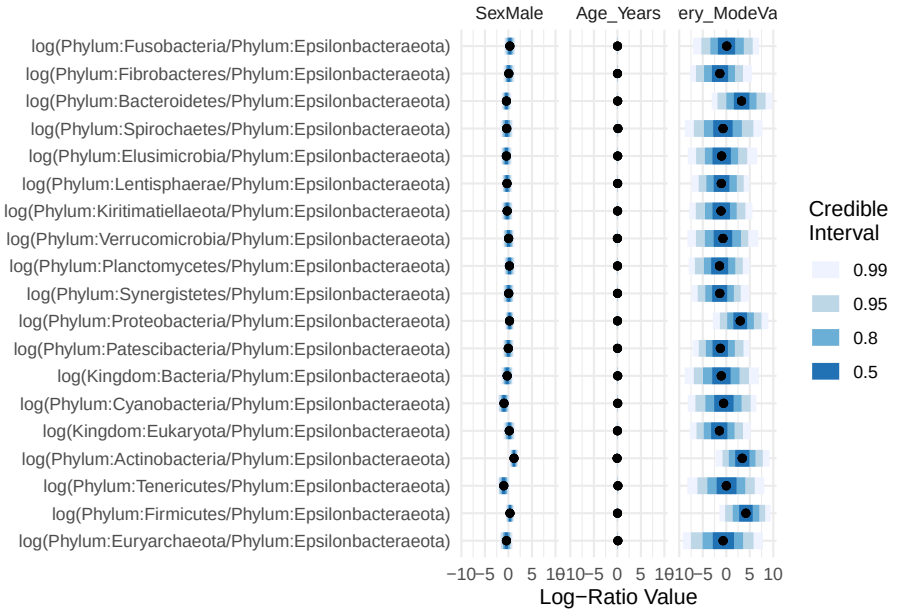
Printing a summary about the posterior:

```
ppc_summary(posterior)
```

```
## Proportions of Observations within 95% Credible Interval: 0.9979296
```

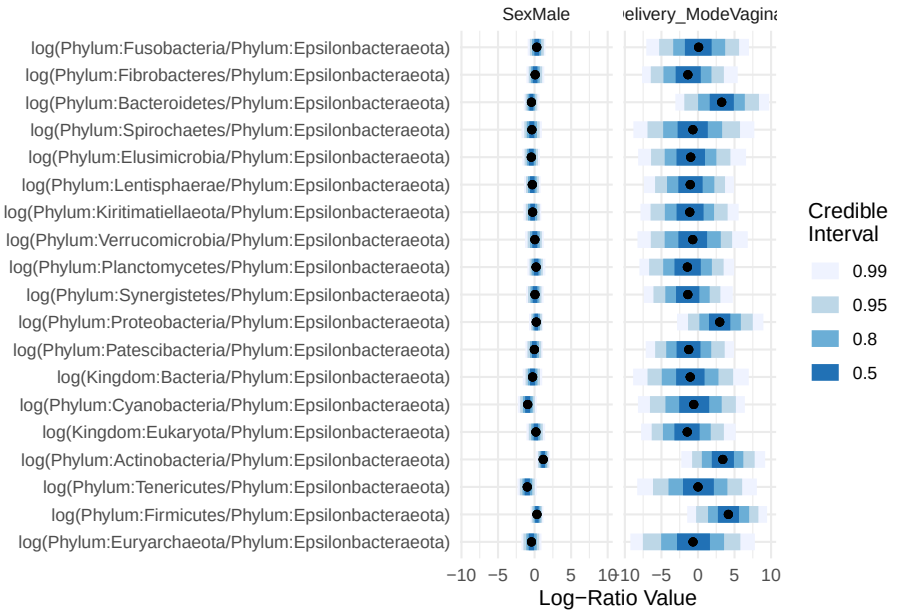
Plotting the summary of the posterior distributions of the regression parameters:

```
names_categories(posterior) <- rownames(Y)
plot(posterior,par="Lambda",focus.cov=rownames(X)[2:4])
```



Taking a closer look at “Sex” and “Delivery_Mode”:

```
plot(posterior, par="Lambda", focus.cov = rownames(X)[c(2,4)])
```



17.3 Interactive 3D Plots

```
# Installing required packages
if (!require(rgl)){
  BiocManager::install("rgl")
}
if (!require(plotly)){
  BiocManager::install("plotly")
}
```

```
library(knitr)
library(rgl)
knitr::knit_hooks$set(webgl = hook_webgl)
```

In this section we make a 3D version of the earlier Visualizing the most dominant genus on PCoA (see 5), with the help of the plotly (Sievert, 2020).

```
# Installing the package
if (!require(curatedMetagenomicData)){
  BiocManager::install("curatedMetagenomicData")
```

```
}
# Importing necessary libraries
library(curatedMetagenomicData)
library(dplyr)
library(DT)
library(mia)
library(scater)

# Querying the data
tse <- sampleMetadata %>%
    filter(age >= 18) %>% # taking only data of age 18 or above
    filter(!is.na(alcohol)) %>% # excluding missing values
    returnSamples("relative_abundance")

tse_Genus <- agglomerateByRank(tse, rank="genus")
tse_Genus <-
 ↪  addPerSampleDominantTaxa(tse_Genus,assay_name="relative_abundance",
 ↪  name = "dominant_taxa")

# Performing PCoA with Bray-Curtis dissimilarity.
tse_Genus <- runMDS(tse_Genus, FUN = vegan::vegdist, ncomponents
 ↪  = 3,
              name = "PCoA_BC", exprs_values =
               ↪  "relative_abundance")

# Getting the 6 top taxa
top_taxa <- getTopTaxa(tse_Genus,top = 6, assay_name =
 ↪  "relative_abundance")

# Naming all the rest of non top-taxa as "Other"
most_abundant <- lapply(colData(tse_Genus)$dominant_taxa,
                  function(x){if (x %in% top_taxa) {x} else
                   ↪  {"Other"}})

# Storing the previous results as a new column within colData
colData(tse_Genus)$most_abundant <- as.character(most_abundant)

# Calculating percentage of the most abundant
most_abundant_freq <- table(as.character(most_abundant))
most_abundant_percent <-
 ↪  round(most_abundant_freq/sum(most_abundant_freq)*100, 1)

# Retrieving the explained variance
e <- attr(reducedDim(tse_Genus, "PCoA_BC"), "eig");
var_explained <- e/sum(e[e>0])*100
```

Interactive 3D visualization of the most dominant genus on PCoA. Note that labels at legend can be used to visualize one or more Genus separately (double click to isolate one from the others, or toggle to select multiple ones).

```
library(plotly)

# 3D Visualization
reduced_data  <- as.data.frame(reducedDim(tse_Genus)[,])
names(reduced_data) <- c("PC1","PC2","PC3")
plot_ly(reduced_data, x=~PC1,y=~PC2,z=~PC3)%>%

  ↪ add_markers(color=sapply(strsplit(colData(tse_Genus)$most_abundant,
  ↪ "_"), tail, 1), size=5,
            colors=c("black", "blue", "lightblue", "darkgray",
             ↪ "magenta", "darkgreen", "red")) %>%
  layout(scene=list(xaxis=list(title = paste("PC1
  ↪ (",round(var_explained[1],1),"%)")),
                    yaxis=list(title = paste("PC2
                     ↪ (",round(var_explained[2],1),"%)")),
                    zaxis=list(title = paste("PC3
                     ↪ (",round(var_explained[3],1),"%)"))))
```

Chapter 18

Acknowledgments

This work would not have been possible without the countless contributions and interactions with other researchers, developers, and users. We express our gratitude to the entire Bioconductor community for developing this high-quality open research software repository for life science analytics, continuously pushing the limits in emerging fields (Gentleman et al., 2004, Huber et al. (2015)). The developers and contributors of this online tutorial are listed in Chapter **??**.

The base ecosystem of data containers, packages, and tutorials was set up as a collaborative effort by Tuomas Borman, Henrik Eckermann, Chouaib Benchraka, Chandler Ross, Shigdel Rajesh, Yağmur Şimşek, Giulio Benedetti, Sudarshan Shetty, Felix Ernst, and Leo Lahti.

The work has been supported by the COST Action network on Statistical and Machine Learning Techniques for Human Microbiome Studies (ML4microbiome) (Moreno-Indias et al., 2021).

The framework is based on the *TreeSummarizedExperiment* data container created by Ruizhu Huang and others (Huang, 2020), and on the MultiAssayExperiment by Marcel Ramos et al. (Ramos et al., 2017). The idea of using these containers as a basis for microbiome data science was initially advanced by the groundwork of Domenick Braccia, Héctor Corrada Bravo and others, and subsequently brought together with other microbiome data science developers (Shetty and Lahti, 2019).

Ample demonstration data resources have been made available as the curatedMetagenomicData project by Edoardo Pasolli, Lucas Schiffer, Levi Waldron and others (Pasolli et al., 2017) adding important support. A number of other contributors have advanced the ecosystem further, and will be acknowledged in the individual packages, pull requests, issues, and other work.

The work has drawn inspiration from many sources, most notably from the work on *phyloseq* by Paul McMurdie and Susan Holmes (McMurdie and Holmes, 2013)

who pioneered the work on rigorous and reproducible microbiome data science ecosystems in R/Bioconductor. The phyloseq framework continues to provide a vast array of complementary packages and methods for microbiome studies, and we aim to support full interoperability.

The open source books by Susan Holmes and Wolfgang Huber, Modern Statistics for Modern Biology (Holmes and Huber, 2019) and by Garret Grolemund and Hadley Wickham, the R for Data Science (Grolemund and Wickham, 2017), and Richard McElreath's Statistical Rethinking and the associated online resources by Solomon Kurz (McElreath, 2020) are key references that advanced reproducible data science training and dissemination. The Orchestrating Single-Cell Analysis with Bioconductor, or *OSCA* book by Robert Amezquita, Aaron Lun, Stephanie Hicks, and Raphael Gottardo (Amezquita et al., 2020b) has implemented closely related work on the *SummarizedExperiment* data container and its derivatives in the field of single cell sequencing studies. Many approaches used in this book have been derived from the OSCA framework, with various adjustments and extensions dedicated to microbiome data science.

Chapter 19

Exercises

Here you can find assignments on different topics.

Tips for exercises:

- Add comments that explain what each line or lines of code do. This helps you and others to understand your code and find bugs. Furthermore, it is easier for you to reuse the code, and it promotes transparency.
- Interpret results by comparing them to literature. List main findings, so that results can easily be understood by others without advanced data analytics knowledge.
- Avoid hard-coding. Use variables which get values in the beginning of the pipeline. That way it is easier for you to modify parameters and reuse the code.

19.1 Workflows

19.1.1 Reproducible reporting

1. Create a new Quarto file

- Rstudio has ready-made templates for this
- Creating Quarto Documents

2. Add a code chunk and name it.
3. Render (or knit) the file into pdf or html format (Hint: Quarto Rstudio rendering)
4. Import e.g., iris dataset, and add a dotplot with a title (Hint: geom_dotplot)

5. Create another code chunk and plot.
6. Adjust figure size and hide the code chunk from output report.

- Sizing
- chunk-options

7. Add some text.
8. Add R commands within the text (Hint: code-chunks)
9. Update HTML file from the qmd file (Hint: Quarto Rstudio rendering)

For tips on Quarto, see Quarto tutorial

19.2 TreeSummarizedExperiment: basic components

You can also check the function reference in the mia package

19.2.1 Basic summaries

1. Load experimental dataset from mia (e.g. `peerj13075` with the `data()` command; see OMA section 2.4 Demonstration Data; also see loading experimental data).
2. Check a summary about the TreeSE object loaded (Hint: `summary()`)
3. What are the dimensions? (How many samples there are, and how many taxa in each taxonomic rank?) (Hint: material in Section 2 may help)
4. List sample and features names for the data (rownames, colnames..)

19.2.2 Taxonomic abundance table (assay)

1. (Load example data)
2. Fetch the list of available assays (Hints: assayNames)
3. Fetch the `counts` assay, and show part of it. (Hint: assay-data)

19.2.3 Sample side information

1. (Load example data)
2. Fetch and show data about samples (Hint: colData)
3. Get abundance data for all taxa for a specific sample (sample names: function `colnames(tse)`)

- example

19.2.4 Feature side information

1. (Load example data)
2. Fetch and show data on the (taxonomic) features of the analyzed samples (Hint: rowData)
3. Get abundance data for all samples given a specific features (Hint: example)

Optional:

4. Create taxonomy tree based on the taxonomy mappings display its information:

- generate a taxonomic tree on the fly
- rowtree

19.2.5 Other components

Try to extract some of the other TreeSE elements. These are not always included:

- Experiment metadata
- Sample tree (colTree)
- Phylogenetic tree (rowTree)
- Feature sequences information (DNA sequence slot)

19.3 TreeSummarizedExperiment: data import

Import data from CSV files to TreeSE (see shared data folder for example data sets).

1. Import the data files in R
2. Construct a TreeSE data object (see Ch. 2)
3. Check that importing is done correctly. E.g., choose random samples and features, and check that their values equal between raw files and TreeSE.

Useful functions: DataFrame, TreeSummarizedExperiment, matrix, rownames, colnames, SimpleList

Optional:

4. Import data from another format (functions: loadFromMetaphlan | loadFromMothur | loadFromQIIME2 | makeTreeSummarizedExperimentFromBiom | makeTreeSummarizedExperimentFromDADA2 …)
5. Try out conversions between TreeSE and phyloseq data containers (makeTreeSummarizedExperimentFromPhyloseq; makephyloseqFromTreeSummarizedExperiment)

19.4 Data manipulation

19.4.1 Subsetting

1. Subset the TreeSE object to specific samples
2. Subset the TreeSE object to specific features
3. Subset the TreeSE object to specific samples and features

19.4.2 Library sizes

1. Calculate library sizes
2. Subsample / rarify the counts (see: subsampleCounts)

Useful functions: nrow, ncol, dim, summary, table, quantile, unique, addPerCellQC, agglomerateByRank

19.4.3 Prevalent and core taxonomic features

1. Estimate prevalence for your chosen feature (row, taxonomic group)
2. Identify all prevalent features and subset the data accordingly
3. Report the thresholds and the dimensionality of the data before and after subsetting
4. Visualize prevalence

Useful functions: getPrevalence, getPrevalentTaxa, subsetByPrevalentTaxa

19.4.4 Explore the data

1. Summarize sample metadata variables. (How many age groups, how they are distributed? 0%, 25%, 50%, 75%, and 100% quantiles of library size?)
2. Create two histograms. Another shows the distribution of absolute counts, another shows how CLR transformed values are distributed.
3. Visualize how relative abundances are distributed between taxa in samples.

Useful functions: nrow, ncol, dim, summary, table, quantile, unique, transformCounts, ggplot, wilcox.test, agglomerateByRank, plotAbundance

19.4.5 Other functions

1. Merge data objects (merge, mergeSEs)
2. Melt the data for visualization purposes (meltAssay)

19.5 Transformations

19.5.1 Transformations

1. Transform abundance data with relative abundance and add a relative abundance assay (see data-transformation)
2. Transform abundance data with clr transformation and add a new assay
3. List the available assays by name
4. Pick one of the assays and show a subset of it
5. Subset the entire TreeSE data object, and check how this affects individual (transformed) assays

Optional:

6. If the data has phylogenetic tree, perform the phILR transformation

19.6 Taxonomic levels

19.6.1 Taxonomic levels

1. List the available taxonomic ranks in the data
2. Merge the data to Phylum level
3. Report dimensionality before and after aggomeration

Optional:

4. Perform CLR transformation on the data; does this affect agglomeration?
5. List full taxonomic information for some given taxa (Hint: mapTaxonomy)

Useful functions: taxonomyRanks, agglomerateByRank, mergeRows

19.6.2 Alternative experiments (altExp)

1. Create taxonomic abundance tables for all different levels (splitByRanks)
2. Check the available alternative experiment (altExp) names before and after splitByRanks
3. Pick specific "experiment" (taxonomic rank) from specific altExp; and a specific assay

Optional:

4. Split the data based on other features (splitOn)

19.7 Alpha diversity

19.7.1 Alpha diversity basics

1. Calculate alpha diversity indices
2. Test if data agglomeration to higher taxonomic ranks affects the indices
3. Look for differences in alpha diversity between groups or correlation with a continuous variable

Useful functions: estimateDiversity, colSums, agglomerateByRank, kruskal.test, cor

19.7.2 Alpha diversity extra

1. Estimate Shannon diversity for the data
2. Try also another diversity index and compare the results with a scatterplot
3. Compare Shannon diversity between groups (boxplot)
4. Is diversity significantly different between vegan and mixed diet?
5. Calculate and visualize library size, compare with diversity

Useful functions: estimateDiversity, colSums, geom_point, geom_boxplot

19.8 Community composition

19.8.1 Beta diversity basics

1. Visualize community variation with different methods (PCA, MDS, NMDS, etc.) with plotReduceDim and with different dissimilarities and transformations,plot also other than the first two axes.

2. Use PERMANOVA to test differences in beta diversity. You can also try including continuous and/or categorical covariates
3. If there are statistical differences in PERMANOVA, test PERMDISP2 (betadisper function)
4. Do clustering
5. Try RDA to test the variance explained by external variables

19.8.2 Beta diversity extra

1. Install the latest development version of mia from GitHub.
2. Load experimental dataset from mia.
3. Create a PCoA with Aitchison dissimilarities. How much coordinate 1 explains the differences? How about coordinate 2?
4. Create dbRDA with Bray-Curtis dissimilarities on relative abundances. Use PERMANOVA. Can differences between samples be explained with variables of sample meta data?
5. Analyze diets' association on beta diversity. Calculate dbRDA and then PERMANOVA. Visualize coefficients. Which taxa's abundances differ the most between samples?
6. Interpret your results. Is there association between community composition and location? What are those taxa that differ the most; find information from literature.

Useful functions: runMDS, runRDA, anova.cca, transformCounts, agglomerateByRank, ggplot, plotReducedDim, vegan::adonis2

19.9 Visualization

19.9.1 Multivariate ordination

1. Load experimental dataset from mia.
2. Create PCoA with Bray-Curtis dissimilarities
3. Create PCA with Aitchison dissimilarities
4. Visualize and compare both
5. Test other transformations, dissimilarities, and ordination methods

Useful functions: runMDS, runNMDS, transformCounts, ggplot, plotReducedDim

19.9.2 Heatmap visualization

1. Load experimental dataset from mia.

2. Visualize abundances with heatmap
3. Visualize abundances with heatmap after CLR + Z transformation

See the OMA book for examples.

19.10 Differential abundance

19.10.1 Univariate analyses

1. Get the abundances for an individual feature (taxonomic group / row)
2. Visualize the abundances per group with boxplot / jitterplot
3. Is the difference significant (Wilcoxon test)?
4. Is the difference significant (linear model with covariates)?
5. How do transformations affect the outcome (log10, clr..)?
6. Get p-values for all features (taxa), for instance with a for loop
7. Do multiple testing correction
8. Compare the results from different tests with a scatterplot

Useful functions: [], ggplot2::geom_boxplot, ggplot2::geom_jitter, wilcox.test, lm.test, transformCounts, p.adjust

19.10.2 Differential abundance analysis

1. install the latest development version of mia from GitHub.
2. Load experimental dataset from mia.
3. Compare abundances of each taxa between groups. First, use Wilcoxon or Kruskall-Wallis test. Then use some other method dedicated to microbiome data.
4. Summarize findings by plotting a taxa vs samples heatmap. Add column annotation that tells the group of each sample, and row annotation that tells whether the difference of certain taxa was statistically significant.
5. Choose statistically significant taxa and visualize their abundances with boxplot & jitterplot.

Useful functions: wilcox.test, kruskal.test, ggplot, pheatmap, ComplexHeatMap::Heatmap, ancombc, aldex2, maaslin2, agglomerateByRank, transformCounts, subsetByPrevalentTaxa

19.11 Multiomics

19.11.1 Introduction to MultiAssayExperiment (MAE)

1. Create TreeSE data containers from individual CSV files.
2. Combine TreeSE into MAE.
3. Check that each individual experiment of MAE equals corresponding TreeSE.
4. Take a subset of MAE (e.g., 10 first samples), and observe the subsetted MAE.

Useful functions: DataFrame, TreeSummarizedExperiment, matrix, rownames, colnames, MultiAssayExperiment, ExperimentList, SimpleList

19.11.2 Introduction to multiomics

1. Load experimental dataset from microbiomeDataSets (e.g., HintikkaXO-Data).
2. Analyze correlations between experiments. (Taxa vs lipids, Taxa vs biomarkers, Lipids vs biomarkers)
3. Agglomerate taxa data.
4. Apply CLR to taxa data, apply log10 to lipids and biomarkers.
5. Perform cross-correlation analyses and visualize results with heatmaps. (Use Spearman coefficients)
6. Is there significant correlations? Interpret your results.

Useful functions: pheatmap, ComplexHeatMap::Heatmap, ggplot, transformCounts, testExperimentCrossAssociation

Bibliography

A, C. (1984). Non-parametric estimation of the number of classes in a population. *Scandinavian Journal of Statistics*, 11(4):265–270.

A, C. and SM, L. (1992). Estimating the number of classes via sample coverage. *Journal of the American statistical Association*, 87(417):210–217.

Amezquita, R., Lun, A., Hicks, S., and Gottardo, R. (2020a). *Orchestrating Single-Cell Analysis with Bioconductor*. Bioconductor.

Amezquita, R. A., Lun, A. T., Becht, E., Carey, V. J., Carpp, L. N., Geistlinger, L., Marini, F., Rue-Albrecht, K., Risso, D., Soneson, C., Waldron, L., Pagès, H., Smith, M. L., Huber, W., Morgan, M., Gottardo, R., and Hicks, S. C. (2020b). Orchestrating single-cell analysis with bioconductor. *Nature Methods*, 17:137–145.

Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, 26(1):32–46.

Anderson, M. J. (2006). Distance-based tests for homogeneity of multivariate dispersions. *Biometrics*, 62:245–253.

Argelaguet, R. e. a. (2018). Multi-omics factor analysis—a framework for unsupervised integration of multi-omics data sets. *Molecular Systems Biology*, 14(6):e8124.

Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., and Holmes, S. P. (2016a). Dada2: High-resolution sample inference from illumina amplicon data. *Nature Methods*, 13:581–583.

Callahan, B. J., Sankaran, K., Fukuyama, J. A., McMurdie, P. J., and Holmes, S. P. (2016b). Bioconductor workflow for microbiome data analysis: from raw reads to community analyses [version 2; peer review: 3 approved]. *F1000Research*, 5:1492.

Chambers, J. (2006). How s4 methods work. Technical report, Technical report.

Chambers, J. M. (2008). *Software for data analysis: programming with R*, volume 2. Springer.

Chen, Y., Lun, A. A. T., and Smyth, G. K. (2016). From reads to genes to pathways: differential expression analysis of rna-seq experiments using rsubread and the edger quasi-likelihood pipeline. *F1000Research*, 5:1438.

Davis, N. M., Proctor, D. M., Holmes, S. P., Relman, D. A., and Callahan, B. J. (2018). Simple statistical identification and removal of contaminant sequences in marker-gene and metagenomics data. *Microbiome*, 6(1):1–14.

Ernst, F. G., Borman, T., and Lahti, L. (2022). *miaViz: Microbiome Analysis Plotting and Visualization*. R package version 1.2.1.

Ernst, F. G., Shetty, S., and Lahti, L. (2020). *mia: Microbiome analysis*. R package version 0.98.15.

Faith, D. P. (1992). Conservation evaluation and phylogenetic diversity. *Biological Conservation*, 61(1):10.

Gentleman, R. (2008). *R programming for bioinformatics*. CRC Press.

Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A. J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J. Y., and Zhang, J. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80.

Gloor, G. B., Macklaim, J. M., and Fernandes, A. D. (2016). Displaying Variation in Large Datasets: Plotting a Visual Summary of Effect Sizes. *Journal of Computational and Graphical Statistics*, 25(3):971–979.

Grolemund, G. and Wickham, H. (2017). *R for Data Science*, volume 77(21); e39-42. O'Reilly.

Gu, Z. (2022). Complex heatmap visualization. *iMeta*, 1(3):e43.

Hintikka, J., Lensu, S., Mäkinen, E., Karvinen, S., Honkanen, M., Lindén, J., Garrels, T., Pekkala, S., and Lahti, L. (2021). Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: Associating taxonomic and metabolomic patterns in fecal microbiomes with biclustering. *International journal of environmental research and public health*, 18(8):4049.

Holmes, S. and Huber, W. (2019). *Modern Statistics for Modern Biology*. Cambridge University Press, New York, NY.

Huang, R. (2020). *TreeSummarizedExperiment: a S4 Class for Data with Tree Structures*. R package version 1.6.2.

Huang, R., Soneson, C., Ernst, F. G., et al. (2021). Treesummarizedexperiment: a s4 class for data with hierarchical structure [version 2; peer review: 3 approved]. *F1000Research*, 9:1246.

Huber, W., Carey, V. J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B. S., Bravo, H. C., Davis, S., Gatto, L., Girke, T., Gottardo, R., Hahne, F., Hansen, K. D., Irizarry, R. A., Lawrence, M., Love, M. I., MacDonald, J., Obenchain, V., Ole's, A. K., Pag'es, H., Reyes, A., Shannon, P., Smyth, G. K., Tenenbaum, D., Waldron, L., and Morgan, M. (2015). Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods*, 12(2):115–121.

Kembel, S. W., Cowan, P. D., Helmus, M. R., Cornwell, W. K., Morlon, H., Ackerly, D. D., Blomberg, S. P., and Webb, C. O. (2010). *Picante: R tools for integrating phylogenies and ecology*. R package version 1.8.2.

Khleborodova, A. (2021). *lefser: R implementation of the LEfSE method for microbiome biomarker discovery*. R package version 1.4.0.

Lahti, L. (2021). *miaTime: time series analysis*. R package version 0.1.0.

Lahti, L., Ernst, F. G., and Shetty, S. (2021a). *microbiomeDataSets: Experiment Hub based microbiome datasets*. R package version 1.2.0.

Lahti, L., JSalojärvi, Salonen, A., Scheffer, M., and de Vos, W. (2014). Tipping elements in the human intestinal ecosystem. *Nature Communications*, 2014:1–10.

Lahti, L., Shetty, S., Ernst, F. M., et al. (2021b). *Orchestrating Microbiome Analysis with Bioconductor [beta version]*.

Lawrence, M., Huber, W., Pagès, H., Aboyoun, P., Carlson, M., Gentleman, R., Morgan, M., and Carey, V. (2013). Software for computing and annotating genomic ranges. *PLoS Computational Biology*, 9.

Lin, H. and Peddada, S. D. (2020a). Analysis of compositions of microbiomes with bias correction. *Nature communications*, 11(1):1–11.

Lin, H. and Peddada, S. D. (2020b). Analysis of compositions of microbiomes with bias correction. *Nat Commun*, 11(1):3514.

Love, M. I., Huber, W., and Anders, S. (2014). Moderated estimation of fold change and dispersion for rna-seq data with deseq2. *Genome Biology*, 15:550.

Lun, A. (2021). *bluster: Clustering Algorithms for Bioconductor*. R package version 1.3.0.

Lun, A. and Risso, D. (2020). *SingleCellExperiment: S4 Classes for Single Cell Data*. R package version 1.12.0.

Mallick, H., Rahnavard, A., and McIver, L. J. (2020). *MaAsLin 2: Multivariable Association in Population-scale Meta-omics Studies*. R/Bioconductor package.

Mandal, S., Van Treuren, W., White, R. A., Eggesbø, M., Knight, R., and Peddada, S. D. (2015). Analysis of composition of microbiomes: A novel method for studying microbial composition. *Microbial Ecology in Health & Disease*, 26(0).

Martin, B. D., Witten, D., and Willis, A. D. (2021). *corncob: Count Regression for Correlated Observations with the Beta-Binomial*. R package version 0.2.0.

McCarthy, D., Campbell, K., Lun, A., and Wills, Q. (2020). *scater: Single-Cell Analysis Toolkit for Gene Expression Data in R*. R package version 1.18.3.

McElreath, R. (2020). *Statistical Rethinking*. Chapman and Hall/CRC. with implementation by Solomon Kurz: https://bookdown.org/ajkurz/Statistical_Rethinking_recoded/.

McInnes, L., Healy, J., and Melville, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv e-prints*, page arXiv:1802.03426.

McMurdie, P. and Holmes, S. (2013). *phyloseq:* an r package for reproducible interactive analysis and graphics of microbiome census data. *PLoS ONE*, 8:e61217.

McMurdie, P. J. and Holmes, S. (2014). Waste not, want not: why rarefying microbiome data is inadmissible. *PLoS computational biology*, 10(4):e1003531.

Moreno-Indias, I., Lahti, L., Nedyalkova, M., Elbere, I., Roshchupkin, G. V., Adilovic, M., Aydemir, O., Bakir-Gungor, B., de Santa Pau, E. C., D'Elia, D., Desai, M. S., Falquet, L., Gundogdu, A., Hron, K., Klammsteiner, T., Lopes, M. B., Zambrano, L. J. M., Marques, C., Mason, M., May, P., Pašić, L., Pio, G., Pongor, S., Promponas, V. J., Przymus, P., Sáez-Rodríguez, J., Sampri, A., Shigdel, R., Stres, B., Suharoschi, R., Truu, J., Truică, C.-O., Vilne, B., Vlachakis, D. P., Yılmaz, E., Zeller, G., Zomer, A., Gómez-Cabrero, D., and Claesson, M. (2021). Statistical and machine learning techniques in human microbiome studies: contemporary challenges and solutions. *Frontiers in Microbiology*, 12:277.

Morgan, M., Obenchain, V., Hester, J., and Pagès, H. (2020). *SummarizedExperiment: SummarizedExperiment container*. R package version 1.20.0.

Morgan, M. and Shepherd, L. (2021). *ExperimentHub: Client to access ExperimentHub resources*. R package version 2.2.0.

Nearing, J. T., Douglas, G. M., Hayes, M. G., MacDonald, J., Desai, D. K., Allward, N., Jones, C. M. A., Wright, R. J., Dhanani, A. S., Comeau, A. M., and Langille, M. G. I. (2022). Microbiome differential abundance methods produce different results across 38 datasets. *Nature Communications*, 13(1):342.

Oksanen, J., Blanchet, F. G., Friendly, M., Kindt, R., Legendre, P., McGlinn, D., Minchin, P. R., O'Hara, R. B., Simpson, G. L., Solymos, P., Stevens, M. H. H., Szoecs, E., and Wagner, H. (2020). *vegan: Community Ecology Package*. R package version 2.5-7.

Pagès, H., Aboyoun, P., Gentleman, R., and DebRoy, S. (2020). *Biostrings: Efficient manipulation of biological strings*. R package version 2.58.0.

Pasolli, E., Schiffer, L., Manghi, P., Renson, A., Obenchain, V., Truong, D., Beghini, F., Malik, F., Ramos, M., Dowd, J., Huttenhower, C., Morgan, M., Segata, N., and L, W. (2017). Accessible, curated metagenomic data through experimenthub. *Nature Methods*, 14:1023–1024.

Paulson, J., Talukder, H., and Bravo, H. (2017). Longitudinal differential abundance analysis of marker-gene surveys using smoothing splines. *biorxiv*.

Pons, P. and Latapy, M. (2006). Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10:191–218.

Quinn, T. P., Gordon-Rodriguez, E., and Erb, I. (2021). A Critique of Differential Abundance Analysis, and Advocacy for an Alternative. *arXiv:2104.07266 [q-bio, stat]*.

Ramos, M., Schiffer, L., Re, A., Azhar, R., Basunia, A., Cabrera, C. R., Chan, T., Chapman, P., Davis, S., Gomez-Cabrero, D., Culhane, A. C., Haibe-Kains, B., Hansen, K., Kodali, H., Louis, M. S., Mer, A. S., Reister, M., Morgan, M., Carey, V., and Waldron, L. (2017). Software for the integration of multi-omics experiments in bioconductor. *Cancer Research*.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., and Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Salosensaari, A., Laitinen, V., Havulinna, A., Méric, G., Cheng, S., Perola, M., Valsta, L., Alfthan, G., Inouye, M., Watrous, J., Long, T., Salido, R., Sanders, K., Brennan, C., Humphrey, G., Sanders, J., Jain, M., Jousilahti, P., Salomaa, V., and Niiranen, T. (2021). Taxonomic signatures of cause-specific mortality risk in human gut microbiome. *Nature Communications*, 12:1–8.

Shetty, S. and Lahti, L. (2019). Microbiome data science. *Journal of Biosciences*, 44:115. Preprint: https://github.com/openresearchlabs/openresearchlabs.github.io/blob/master/public/publication_resources/pape Shetty-MDS.pdf.

Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC.

Silverman, J. D., Washburne, A. D., Mukherjee, S., and David, L. A. (2017). A phylogenetic transform enhances analysis of compositional microbiota data. *eLife*, 6.

Simsek, Y., Lahti, L., Garza, D., and Faust, K. (2021). miasim r package. Version 1.0.0.

Sprockett, D. D., Martin, M., Costello, E. K., Burns, A. R., Holmes, S. P., Gurven, M. D., and Relman, D. A. (2020). Microbiota assembly, structure, and dynamics among Tsimane horticulturalists of the Bolivian Amazon. *Nat Commun*, 11(1):3772.

Vatanen, T., Kostic, A. D., d'Hennezel, E., Siljander, H., Franzosa, E. A., Yassour, M., Kolde, R., Vlamakis, H., Arthur, T. D., Hämäläinen, A.-M., Peet, A., Tillmann, V., Uibo, R., Mokurov, S., Dorshakova, N., Ilonen, J., Virtanen, S. M., Szabo, S. J., Porter, J. A., Lähdesmäki, H., Huttenhower, C., Gevers, D., Cullen, T. W., Knip, M., , and Xavier, R. J. (2016). Variation in microbiome lps immunogenicity contributes to autoimmunity in humans. *Cell*, 165:842–853.

W, K. S., D, C. P., R, H. M., K, C. W., Helene, M., D, A. D., P, B. S., and O, W. C. (2010). Picante: R tools for integrating phylogenies and ecology. *Bioinformatics*, 26(11):1463–1464.

Whittaker, R. H. (1960). Vegetation of the siskiyou mountains, oregon and california. *Ecological Monographs*, 30(3):279–338.

Wright, E. (2020). *DECIPHER: Tools for curating, analyzing, and manipulating biological sequences*. R package version 2.18.1.

Xie, Y., Dervieux, C., and Riederer, E. (2020). *R Markdown Cookbook*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 9780367563837.

Zhou, H., He, K., Chen, J., and Zhang, X. (2022). LinDA: Linear models for differential abundance analysis of microbiome compositional data. *Genome Biology*, 23(1):95.