Orchestrating Microbiome Analysis with Bioconductor

Authors: Leo Lahti [aut], Tuomas Borman [aut, cre], Felix GM Ernst [aut], and others (see the full list of contributors) [ctb]
Version: 0.98.16 **Modified:** 2023-07-29 **Compiled:** 2023-07-30
Environment: R version 4.3.0 (2023-04-21), Bioconductor 3.17

https://github.com/microbiome/OMA

Contents

Welcome

You are reading the online book, **Orchestrating Microbiome Analysis with 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 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 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

Packages

The Bioconductor microbiome data science framework consists of:

- **data containers**, designed to organize multi-assay microbiome data
- **R/Bioconductor packages** that provide dedicated methods
- **community** of users and developers

This section provides an overview of the package ecosystem. Section 3.3 links to various open microbiome data resources that support this framework.

2.1 Package installation

You can install all packages that are required to run every example in this book via the following command:

```
source("https://raw.githubusercontent.com/microbiome/OMA/master/install_packages.R")
```

2.1.1 Installing specific packages

You can install R packages of your choice with the following command line procedure.

Bioconductor release version is the most stable and tested version but may miss some of the latest methods and updates. It can be installed with:

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

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

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

Github development version provides access to the latest but potentially unstable features. This is useful when you want access to all available tools.

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

2.2 Package ecosystem

Methods for `(Tree)SummarizedExperiment` and `MultiAssayExperiment` data containers are provided by multiple independent developers through R/Bioconductor packages. Some of these are listed below (tips on new packages are welcome).

2.2.1 mia package family

The mia package family provides general methods for microbiome data wrangling, analysis and visualization.

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

2.2.2 Differential abundance

The following DA methods support `(Tree)SummarizedExperiment`.

- ANCOMBC for differential abundance analysis
- benchdamic for benchmarking differential abundance methods
- ALDEx2 for differential abundance analysis

2.2.3 Other packages

- philr (Silverman et al. (2017)) phylogeny-aware phILR transformation

- MicrobiotaProcess for "tidy" analysis of microbiome and other ecological data
- Tools for Microbiome Analysis site listed over 130 R packages for microbiome data science in

2023. Many of these are not in Bioconductor, or do not directly support the data containers used in this book but can be often used with minor modifications. The phyloseq-based tools can be used by converting the TreeSE data into phyloseq with `makePhyloseqFromTreeSummarizedExperiment`.

2.2.4 Open microbiome data

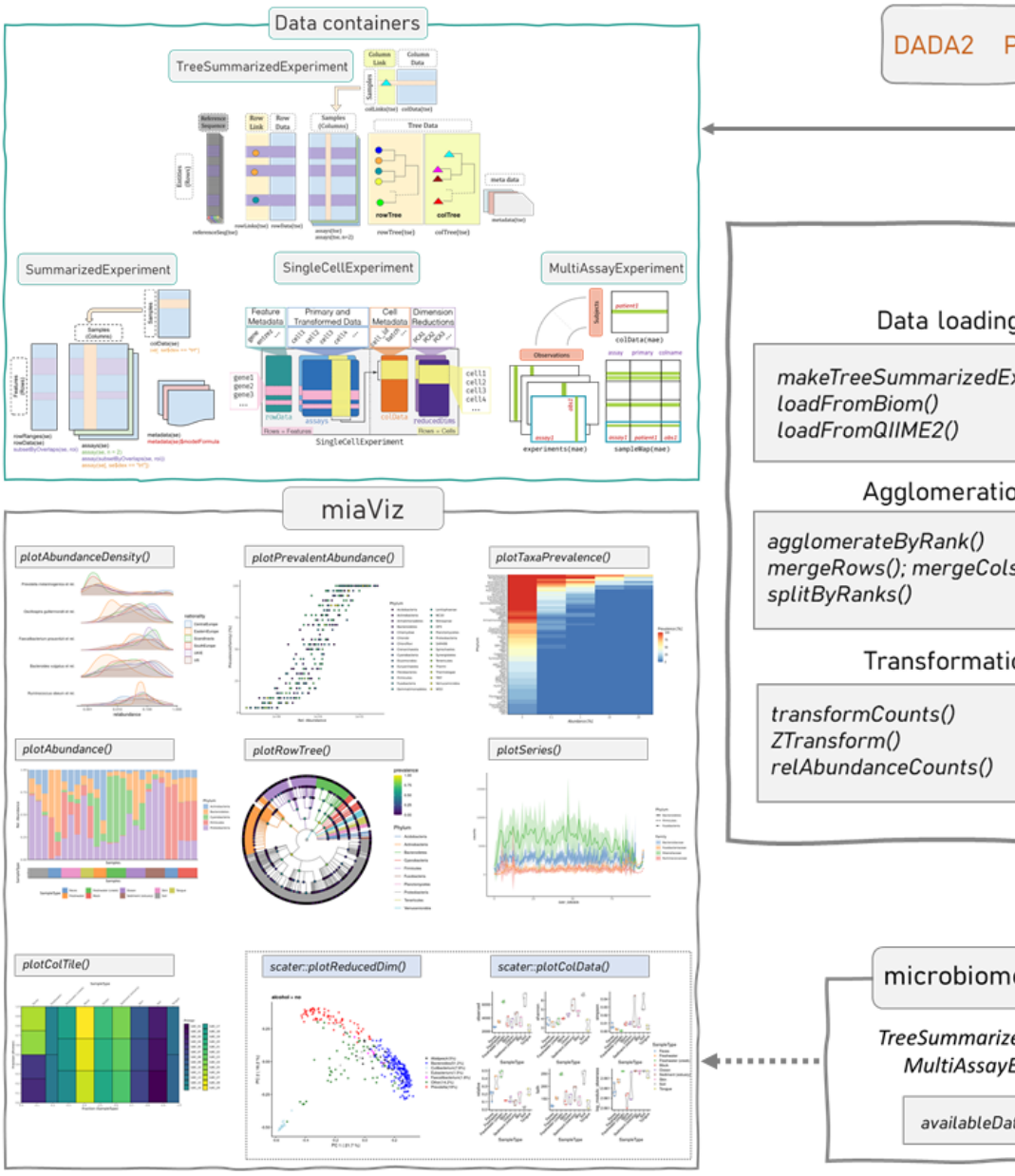
Hundreds of published microbiome data sets are readily available in these data containers (see 3.3).

Chapter 3

Microbiome Data

3.1 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 3.3, and **online tutorials** including this online book as well as the various package vignettes and other materials.



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

3.2 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 containers together in a single object. For example, we can bind microbiome data (in `TreeSE` container) with metabolomic profiling data (in `SE`) container, with (partially) 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. We highlight how the methods used througout in this book relate to this data framework by using the `TreeSummarizedExperiment`, `MultiAssayExperiment`, 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 3.1.

3.2.1 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***.

Let us load example data and rename it as tse.

```
library(mia)
data(hitchip1006, package="miaTime")
tse <- hitchip1006
```

The `assays` slot contains the experimental data as multiple count matrices. The result of `assays` is 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]
```

```
##                               Sample-1 Sample-2 Sample-3 Sample-4 Sample-5
## Actinomycetaceae                     0        0        0        0        0
## Aerococcus                           0        0        0        0        0
## Aeromonas                            0        0        0        0        0
## Akkermansia                         21       36      475       61       34
## Alcaligenes faecalis et rel.         1        1        1        2        1
##                               Sample-6 Sample-7
## Actinomycetaceae                     0        0
## Aerococcus                           0        0
## Aeromonas                            0        0
## Akkermansia                         14       27
## Alcaligenes faecalis et rel.         1        1
```

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

```
tse <- transformAssay(tse, assay.type = "counts", method =
 ↪  "relabundance")
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]
```

```
##                                Sample-1  Sample-2  Sample-3  Sample-4  Sample-5
## Actinomycetaceae             0.0000000 0.000e+00 0.0000000 0.0000000 0.000e+00
## Aerococcus                   0.0000000 0.000e+00 0.0000000 0.0000000 0.000e+00
## Aeromonas                    0.0000000 0.000e+00 0.0000000 0.0000000 0.000e+00
## Akkermansia                  0.0027657 3.547e-03 0.0666106 0.0056195 2.833e-03
```

```
## Alcaligenes faecalis et rel. 0.0001317 9.854e-05 0.0001402 0.0001842 8.333e-05
##                               Sample-6  Sample-7
## Actinomycetaceae             0.0000000 0.0000000
## Aerococcus                   0.0000000 0.0000000
## Aeromonas                    0.0000000 0.0000000
## Akkermansia                  0.0017690 0.0045570
## Alcaligenes faecalis et rel. 0.0001264 0.0001688
```

Here the dimension of the count data remains unchanged in transformation. This is in fact, a requirement for the assays.

3.2.2 colData

`colData` contains data on the samples.

```
colData(tse)
```

```
## DataFrame with 1151 rows and 10 columns
##                    age      sex nationality DNA_extraction_method  project
##              <integer> <factor>    <factor>              <factor> <factor>
## Sample-1            28   male            US                    NA        1
## Sample-2            24   female          US                    NA        1
## Sample-3            52   male            US                    NA        1
## Sample-4            22   female          US                    NA        1
## Sample-5            25   female          US                    NA        1
## ...                ...      ...         ...                   ...      ...
## Sample-1168         50   female Scandinavia                     r       40
## Sample-1169         31   female Scandinavia                     r       40
## Sample-1170         31   female Scandinavia                     r       40
## Sample-1171         52   male   Scandinavia                     r       40
## Sample-1172         52   male   Scandinavia                     r       40
##              diversity   bmi_group  subject      time      sample
##              <numeric>    <factor> <factor> <numeric> <character>
## Sample-1          5.76 severeobese        1         0    Sample-1
## Sample-2          6.06 obese              2         0    Sample-2
## Sample-3          5.50 lean               3         0    Sample-3
## Sample-4          5.87 underweight        4         0    Sample-4
## Sample-5          5.89 lean               5         0    Sample-5
## ...                ...         ...      ...       ...         ...
## Sample-1168       5.87 severeobese      244       8.1 Sample-1168
## Sample-1169       5.87 overweight       245       2.3 Sample-1169
## Sample-1170       5.92 overweight       245       8.2 Sample-1170
## Sample-1171       6.04 overweight       246       2.1 Sample-1171
## Sample-1172       5.74 overweight       246       7.9 Sample-1172
```

3.2.3 rowData

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

```
rowData(tse)
```

```
## DataFrame with 130 rows and 3 columns
##                                       Phylum          Family
##                                  <character>     <character>
## Actinomycetaceae              Actinobacteria  Actinobacteria
## Aerococcus                        Firmicutes         Bacilli
## Aeromonas                     Proteobacteria  Proteobacteria
## Akkermansia                  Verrucomicrobia Verrucomicrobia
## Alcaligenes faecalis et rel.  Proteobacteria  Proteobacteria
## ...                                      ...             ...
## Vibrio                        Proteobacteria  Proteobacteria
## Weissella et rel.                 Firmicutes         Bacilli
## Wissella et rel.                  Firmicutes         Bacilli
## Xanthomonadaceae              Proteobacteria  Proteobacteria
## Yersinia et rel.              Proteobacteria  Proteobacteria
##                                          Genus
##                                    <character>
## Actinomycetaceae                Actinomycetaceae
## Aerococcus                           Aerococcus
## Aeromonas                             Aeromonas
## Akkermansia                          Akkermansia
## Alcaligenes faecalis et rel. Alcaligenes faecalis..
## ...                                         ...
## Vibrio                                   Vibrio
## Weissella et rel.                Weissella et rel.
## Wissella et rel.                  Wissella et rel.
## Xanthomonadaceae                  Xanthomonadaceae
## Yersinia et rel.                  Yersinia et rel.
```

3.2.4 rowTree

Phylogenetic trees also play an important role in the microbiome field. The `TreeSummarizedExperiment` class can keep track of features and node relations via two functions, `rowTree` and `rowLinks`.

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

```
rowTree(tse)
```

```
## NULL
```

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

```
rowLinks(tse)
```

```
## NULL
```

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.

3.2.5 Alternative experiments

Alternative experiments complement *assays*. 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. Another common use case is including abundance tables for different taxonomic ranks. Such alternative experiments concerning the same set of samples can be stored as

1. Separate *assays* assuming that the taxonomic information can be mapped between features 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.

The following shows 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 match.

Let us first agglomerate the data to Phylum level. This yields a new TreeSE data object.

```
tse_phylum <- agglomerateByRank(tse, "Phylum", na.rm=TRUE)
# Both have the same number of columns (samples)
dim(tse)
```

```
## [1]  130 1151
```

```
dim(tse_phylum)
```

```
## [1]    8 1151
```

Then we can add the new phylum-level data object as an alternative experiment in the original data.

```
# Add the new data object to the original data object as an
 ↪  alternative experiment with the name "Phylum"
altExp(tse, "Phylum") <- tse_phylum

# Check the alternative experiment names available in the data
altExpNames(tse)
```

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

We can now subset the data, for instance, and this acts on both altExp and assay data.

```
tse[,1:10]
```

```
## class: TreeSummarizedExperiment
## dim: 130 10
## metadata(0):
## assays(2): counts relabundance
## rownames(130): Actinomycetaceae Aerococcus ... Xanthomonadaceae
##   Yersinia et rel.
## rowData names(3): Phylum Family Genus
## colnames(10): Sample-1 Sample-2 ... Sample-9 Sample-10
## colData names(10): age sex ... time sample
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(1): Phylum
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

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

```
## [1]  8 10
```

For more details on *altExp*, you can check the introduction to the `SingleCellExperiment` package (Lun and Risso, 2020).

3.2.6 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 objects, including `SummarizedExperiment` objects, and the number of samples can differ between the elements.

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

Table 3.1: **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 libraryd 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

3.3 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.

3.3.1 Package data

The `mia` R package contains example datasets 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
```

3.3.1.1 HintikkaXOData

HintikkaXOData is derived from a study about the effects of fat diet and prebiotics on the microbiome of rat models (Hintikka et al., 2021). It is available in the MAE data container for R. The dataset is briefly summarized in these slides.

3.3.2 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 datasets 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* dataset:

```
# 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, package = "mia")
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.

3.3.3 Curated metagenomic data

curatedMetagenomicData is a large collection of curated human microbiome datasets, provided as `(Tree)SummarizedExperiment` objects (Pasolli et al., 2017). The 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)
```

3.3.4 Other data sources

The current collections provide access to vast microbiome data resources. The output has to be converted into TreeSE/MAE separately.

- MGnifyR provides access to EBI/MGnify
- qiitr provides access to QIITA

3.4 Loading experimental microbiome data

3.4.1 16S workflow

Result of amplicon sequencing is a 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, here. After the DADA2 portion of the tutorial is completed, the data is stored into *phyloseq* object (Bonus: Handoff 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)
library(BiocManager)
library(Biostrings)
```

Create arbitrary example sample metadata like it was done in the 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 a *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)
```

3.4.2 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. Some datasets and raw files to learn how to import raw data and construct TreeSE/MAE containers are available in the microbiome data repository.

3.4.2.1 CSV import

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. You can also check the example files and construct your own CSV files accordingly.

Recommendations for the CSV files are the following. File names are arbitrary; we refer here to the same names as in the examples:

- Abundance table (`assay_taxa.csv`): data matrix (features x samples); first column provides feature IDs, the first row provides sample IDs; other values should be numeric (abundances).
- Row data (`rowdata_taxa.csv`): data table (features x info); first column provides feature IDs, the first row provides column headers; this file usually contains the taxonomic mapping between different taxonomic levels. Ideally, the feature IDs (row names) match one-to-one with the abundance table row names.
- Column data (`coldata.csv`): data table (samples x info); first column provides sample IDs, the first row provides column headers; this file usually contains the sample metadata/phenodata (such as subject age, health etc). Ideally, the sample IDs match one-to-one with the abundance table column names.

After you have set up the CSV files, you can read them in R:

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

# Load files
counts  <- read.csv(count_file, row.names=1)   # Abundance table
 ↪  (e.g. ASV data; to assay data)
tax     <- read.csv(tax_file, row.names=1)     # Taxonomy table
 ↪  (to rowData)
samples <- read.csv(sample_file, row.names=1)  # Sample data (to
 ↪  colData)
```

After reading the data in R, ensure the following:

- abundance table (`counts`): numeric `matrix`, with feature IDs as rownames and sample IDs as column names
- rowdata (`tax`): `DataFrame`, with feature IDs as rownames. If this is a `data.frame` you can use the function `DataFrame()` to change the format. Column names are free but in microbiome analysis they usually they refer to taxonomic ranks. The rownames in rowdata should match with rownames in abundance table.
- coldata (`samples`): `DataFrame`, with sample IDs as rownames. If this is a `data.frame` you can use the function `DataFrame()` to change the format. Column names are free. The rownames in coldata should match with colnames in abundance table.

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

Also ensure that the row and column names match one-to-one between abundance table, rowdata, and coldata:

```
# Match rows and columns
counts <- counts[rownames(tax), rownames(samples)]

# Let us ensure that the data is in correct (numeric matrix)
 ↪  format:
counts <- as.matrix(counts)
```

If you hesitate about the format of the data, you can compare to one of the available demonstration datasets, and make sure that your data components have the same format.

There are many different source files and many different ways to read data in R. One can do data manipulation in R as well. Investigate the entries as follows.

```
# coldata rownames match assay colnames
all(rownames(samples) == colnames(counts)) # our dataset
```

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

```
class(samples) # should be data.frame or DataFrame
```

```
## [1] "data.frame"
```

```
# rowdata rownames match assay rownames
all(rownames(tax) == rownames(counts)) # our dataset
```

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

```
class(tax) # should be data.frame or DataFrame
```

```
## [1] "data.frame"
```

```
# Counts
class(counts) # should be a numeric matrix
```

```
## [1] "matrix" "array"
```

3.4.3 Constructing TreeSummarizedExperiment

Now let us create the TreeSE object from the input data tables. Here we also convert the data objects in their preferred formats:

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

The `SimpleList` could be used to include multiple alternative assays, if necessary.

```
# Create a TreeSE
tse_taxa <- TreeSummarizedExperiment(assays =  SimpleList(counts
 ↪  = counts),
                                     colData =
                                      ↪  DataFrame(samples),
                                     rowData = DataFrame(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
```

Now you should have a ready-made TreeSE data object that can be used in downstream analyses.

3.4.4 Constructing MultiAssayExperiment

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, row.names=1)
samples <- read.csv(sample_file, row.names=1)

# Create a TreeSE for the metabolite data
tse_metabolite <- TreeSummarizedExperiment(assays =
 ↪  SimpleList(concs = as.matrix(counts)),
                                            colData =
                                             ↪  DataFrame(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
```

3.4.5 Import functions for standard formats

Specific import functions are provided for:

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

3.4.5.1 Biom import

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 dataset 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.

```
assay(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))
```

3.4.6 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

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 <- transformAssay(tse, MARGIN = "samples",
 ↪  method="relabundance")
molten_tse <- mia::meltAssay(tse,
                        add_row_data = TRUE,
                        add_col_data = TRUE,
                        assay.type = "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 datasets, 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[ , 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 Add or modify data

The information contained by the `colData` of a `TreeSE` can be modified by accessing the desired variables.

```
# modify the Description entries
colData(tse)$Description <- paste(colData(tse)$Description,
 ↪  "modified description")

# view modified variable
head(tse$Description)
```

```
## [1] "Calhoun South Carolina Pine soil, pH 4.9 modified description"
## [2] "Cedar Creek Minnesota, grassland, pH 6.1 modified description"
## [3] "Sevilleta new Mexico, desert scrub, pH 8.3 modified description"
## [4] "M3, Day 1, fecal swab, whole body study modified description"
## [5] "M1, Day 1, fecal swab, whole body study  modified description"
## [6] "M3, Day 1, right palm, whole body study modified description"
```

New information can also be added to the experiment by creating a new variable.

```
# simulate new data
new_data <- runif(ncol(tse))

# store new data as new variable in colData
colData(tse)$NewVariable <- new_data

# view new variable
head(tse$NewVariable)
```

```
## [1] 0.20639 0.42847 0.84933 0.31360 0.09938 0.27090
```

4.3 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(8): X.SampleID Primer ... Description NewVariable
## 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 ... 239967 254851
## rowData names(7): Kingdom Phylum ... Genus Species
## colnames(2): CL3 M31Fcsw
## colData names(8): X.SampleID Primer ... Description NewVariable
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
```

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

4.3.1 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 dataset 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)
```

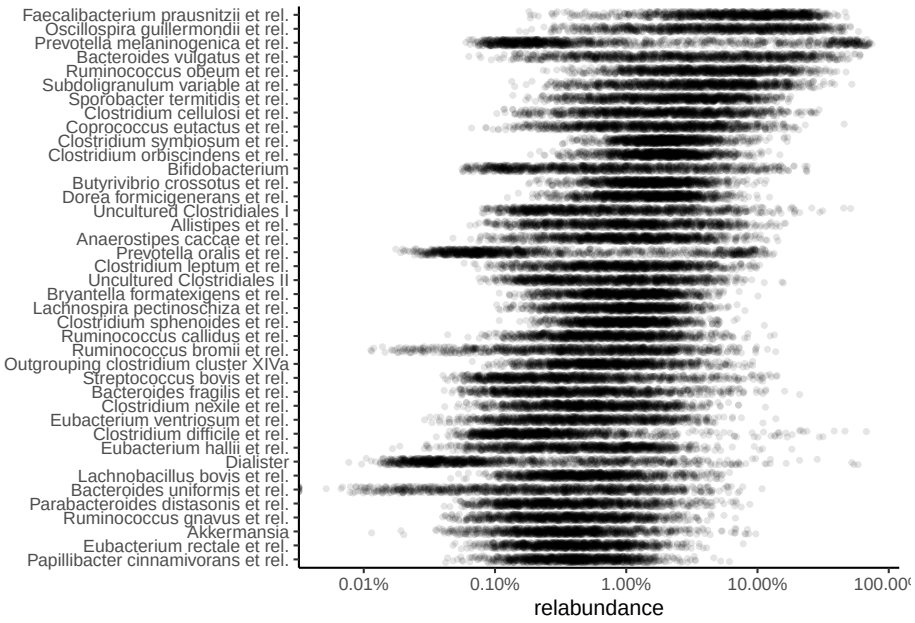
5.1 Abundance

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

Next, a 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:

```
# Load example data
library(miaTime)
```

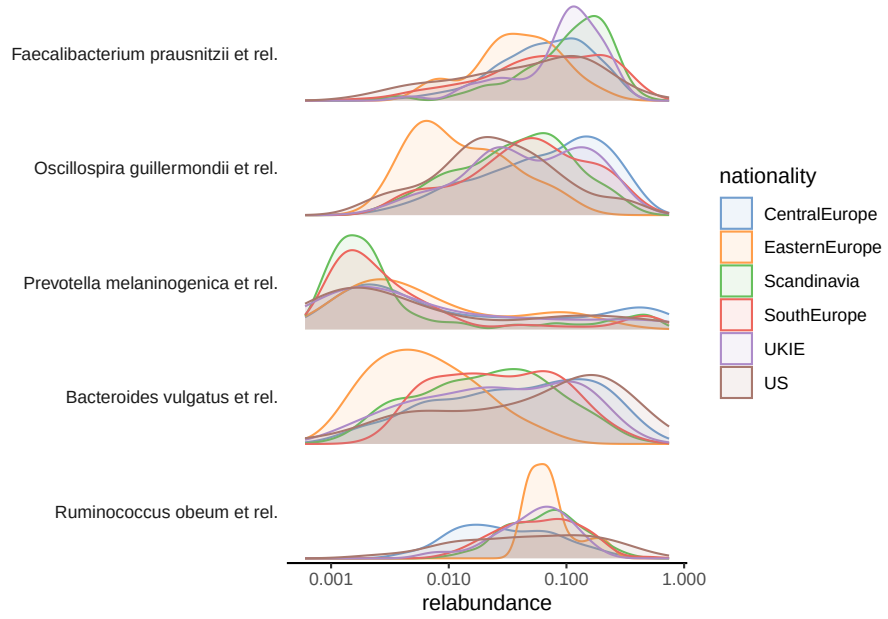
```
library(miaViz)
data(hitchip1006)
tse <- hitchip1006

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

# Use argument names
# assay.type / assay.type / assay.type
# depending on the mia package version
plotAbundanceDensity(tse, layout = "jitter", assay.type =
 ↪  "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.type =
 ↪  "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.

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.type =
 ↪  "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.type = "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.type = "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, `getPrevalentFeatures` is available. This returns the taxa that exceed the given prevalence and detection thresholds.

```
getPrevalentFeatures(tse, detection = 0, prevalence = 50/100)
prev <- getPrevalentFeatures(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`, `getRareFeatures`, `subsetByRareFeatures`).

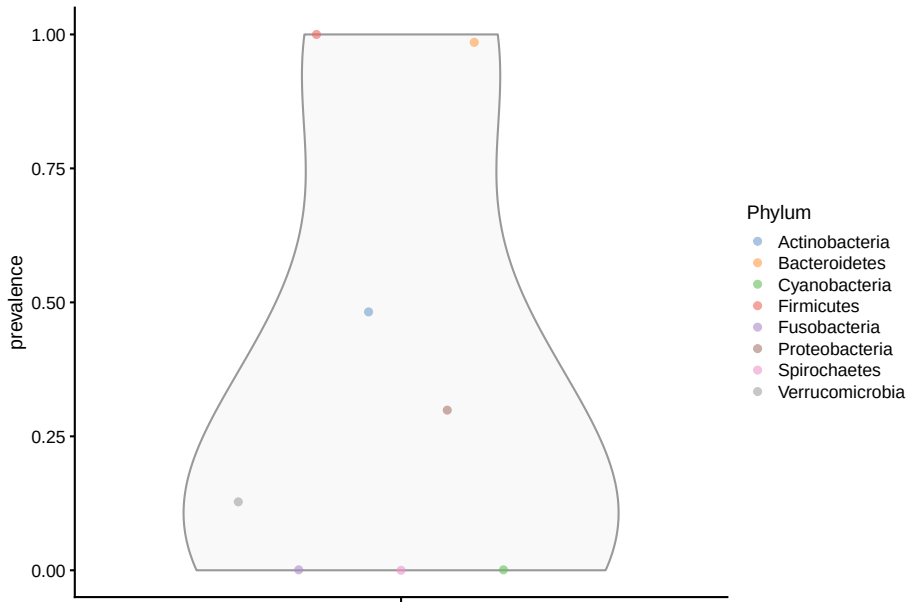
5.2.3 Plotting prevalence

To plot the prevalence, add the prevalence of each taxon to `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.type = "counts", as_relative = TRUE)
```

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

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



The prevalence can also be 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.type = "counts",
                                 ↪ as_relative = TRUE)
```

```
            y
        })
top_phyla <- getTopFeatures(altExp(tse,"Phylum"),
                        method="prevalence",
                        top=5L,
                        assay.type="counts")
top_phyla_mean <- getTopFeatures(altExp(tse,"Phylum"),
                             method="mean",
                             top=5L,
                             assay.type="counts")
x <- unsplitByRanks(tse, ranks = taxonomyRanks(tse)[1:6])
x <- addTaxonomyTree(x)
```

After some preparation, the data is assembled and can be plotted with `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` dataset to illustrate standard microbiome data summaries.

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

5.3.1 Top taxa

The `getTopFeatures` identifies top taxa in the data.

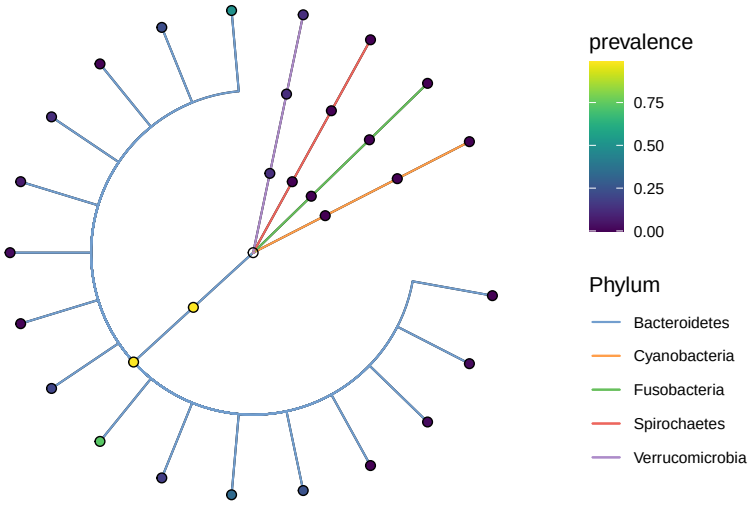


Figure 5.1: Prevalence of top phyla as judged by prevalence

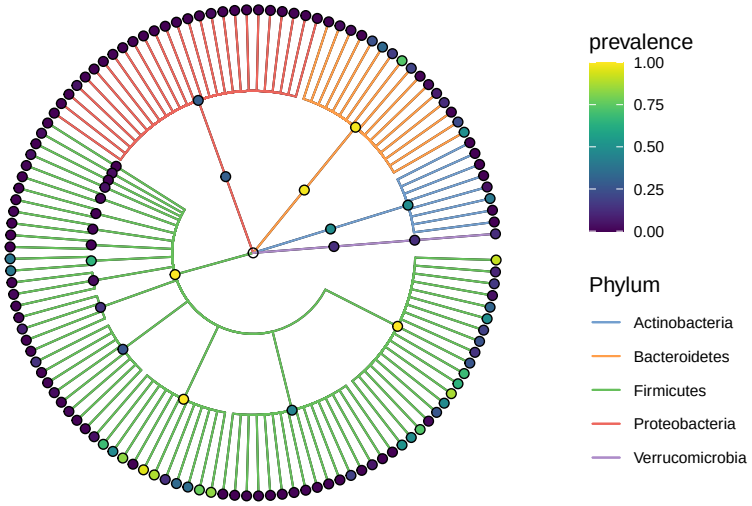


Figure 5.2: Prevalence of top phyla as judged by mean abundance

```
# Pick the top taxa
top_features <- getTopFeatures(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 `perCellQCMetrics`/`addPerCellQC` from the `scater` package. The former one just calculates the values, whereas the latter one directly adds them to `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
```

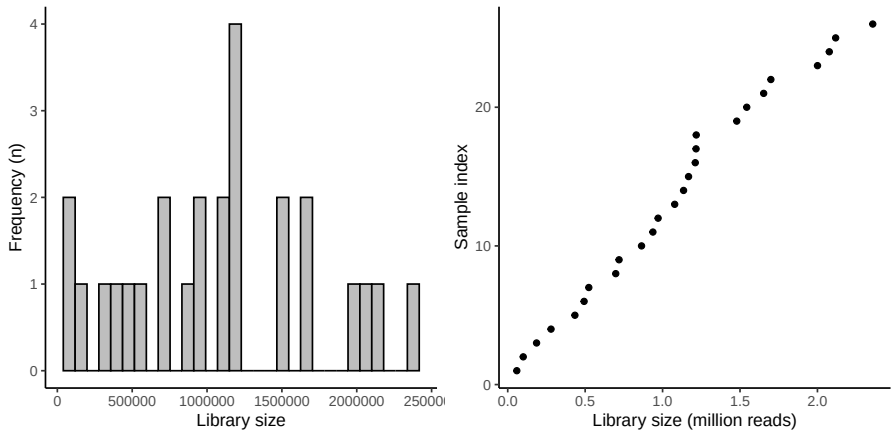


Figure 5.3: Library size distribution.

Library sizes other variables from `colData` can be 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)) %>%
            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")
```

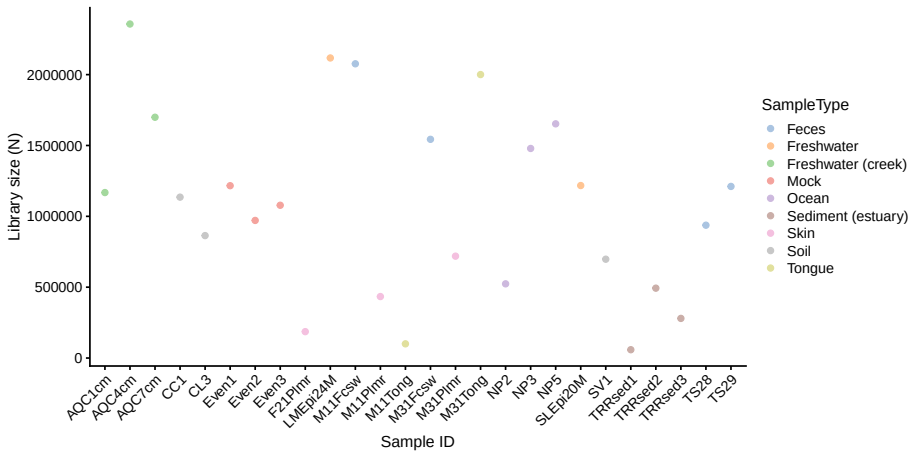


Figure 5.4: Library sizes per sample.

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

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.

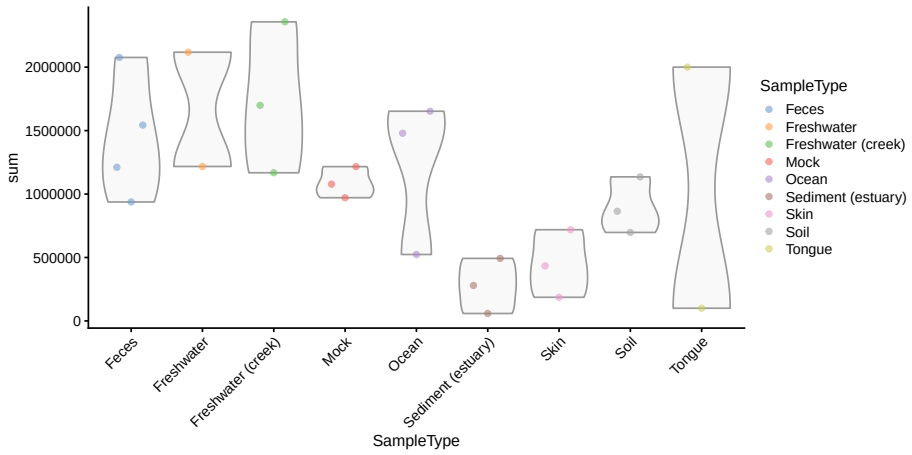


Figure 5.5: Library sizes per sample type.

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

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

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

In this chapter, we will refer to co-abundant groups as CAGs, which are clusters of taxa that co-vary across samples.

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, logical(1))
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 `getUniqueFeatures` gives a list of unique taxa for the specified taxonomic rank.

```
head(getUniqueFeatures(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 <- transformAssay(tse, assay.type = "counts", method =
 ↪  "relabundance")
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)`.

Rare taxa can also be aggregated into a single group "Other" instead of filtering them out. A suitable function for this is `agglomerateByPrevalence`. The number of rare taxa is higher on the species level, which causes the need for data agglomeration by prevalence.

```
altExp(tse, "Species_byPrevalence") <-
  agglomerateByPrevalence(tse,
                          rank
                          =
                          "Species",

                          other_label
                          =
                          "Other",

                          prevalence
                          =
                          5
                          /
                          100,
```

```
                                                                  ↪ detection
                                                                  ↪ =
                                                                  ↪ 1
                                                                  ↪ /
                                                                  ↪ 100,
                                                                  ↪
                                                                  ↪ as_relative
                                                                  ↪ =
                                                                  ↪ T)
altExp(tse, "Species_byPrevalence")
```

```
## class: TreeSummarizedExperiment
## dim: 92 26
## metadata(2): agglomerated_by_rank agglomerated_by_rank
## assays(2): counts relabundance
## rownames(92): pIVWA5 SCA1145 ... Desulfitobacteriumhafniense Other
## 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: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

```
assay(altExp(tse, "Species_byPrevalence"), "relabundance")[88:92,
 ↪  1:7]
```

```
##                                   CL3       CC1       SV1   M31Fcsw   M11Fcsw
## Streptococcusthermophilus   5.787e-06 2.290e-05 1.290e-05 6.032e-04 1.122e-04
## Mitsuokellamultacida        8.101e-06 7.046e-06 1.147e-05 6.479e-07 9.632e-07
## Veillonellaparvula          1.736e-05 1.673e-05 1.720e-05 7.645e-05 1.589e-05
## Desulfitobacteriumhafniense 1.620e-05 1.585e-05 8.602e-06 1.296e-06 9.632e-07
## Other                       8.622e-03 6.787e-03 4.325e-02 2.763e-02 2.682e-03
##                               M31Plmr  M11Plmr
## Streptococcusthermophilus   1.225e-02 0.002478
## Mitsuokellamultacida        2.782e-06 0.000000
## Veillonellaparvula          2.075e-02 0.001143
## Desulfitobacteriumhafniense 0.000e+00 0.000000
## Other                       7.077e-02 0.070752
```

```
# Saving the tse for later
tseGlobalPatterns <- tse
```

6.3.1 Taxa clustering

Another way to agglomerate the data is to cluster the taxa. To do so, we usually start by doing a compositionality aware transformation such as CLR, followed by the application of a standard clustering method.

Here is an example that does a CLR transformation followed by the hierarchical clustering algorithm.

First, we import the library `bluster` that simplifies the clustering.

```
library(bluster)
```

Then we do the CLR transform followed by the clustering. We will cluster with two different distances: the euclidean distance and the kendall distance.

```
# Get the data
data("peerj13075", package = "mia")
tse <- peerj13075

# The result of the CLR transform is stored in the assay clr
tse <- transformAssay(tse, method = "clr", pseudocount = 1)

tse <- transformAssay(tse, assay.type = "clr", method = "z",
                      MARGIN = "features")

# Cluster (with euclidean distance) on the features of the z
 ↪  assay
tse <- cluster(tse,
               assay.type = "z",
               clust.col = "hclustEuclidean",
           MARGIN = "features",
               HclustParam(dist.fun = stats::dist, method =
                ↪  "ward.D2"))

# Declare the Kendall dissimilarity computation function
kendall_dissimilarity <- function(x) {
    as.dist(1 - cor(t(x), method = "kendall"))
}

# Cluster (with Kendall dissimilarity) on the features of the z
 ↪  assay
```

```
tse <- cluster(tse,
               assay.type = "z",
               clust.col = "hclustKendall",
               MARGIN = "features",
               HclustParam(dist.fun = kendall_dissimilarity,
                ↪  method = "ward.D2"))
```

Let us store the resulting cluster indices in the `rowData` column specified with the `clust.col` parameter.

```
# Checking the clusters
clusters_euclidean <- rowData(tse)$hclustEuclidean
head(clusters_euclidean, 10)
```

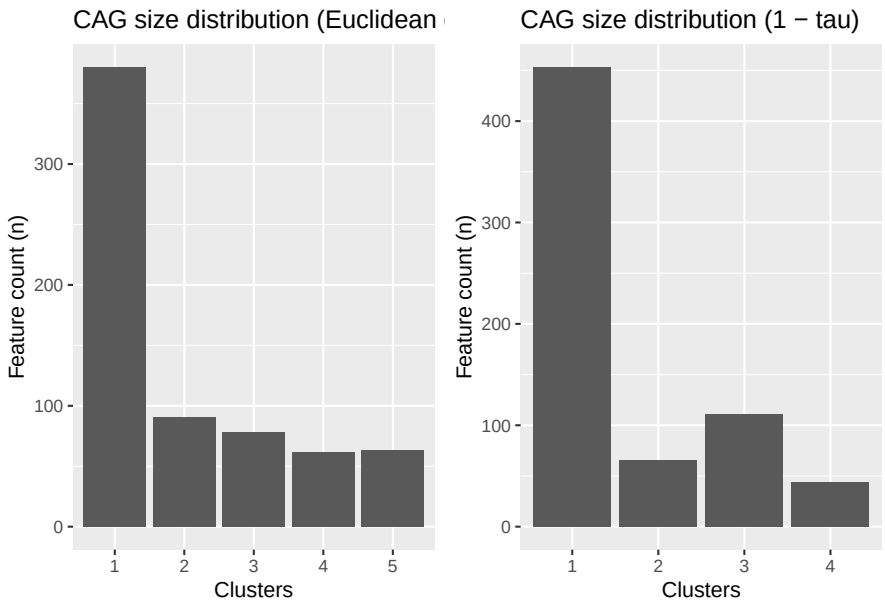
```
##  OTU1  OTU2  OTU7  OTU9 OTU10 OTU12 OTU14 OTU15 OTU18 OTU19
##     1     2     1     1     1     1     3     4     3     2
## Levels: 1 2 3 4 5
```

```
clusters_kendall <- rowData(tse)$hclustKendall
head(clusters_kendall, 10)
```

```
##  OTU1  OTU2  OTU7  OTU9 OTU10 OTU12 OTU14 OTU15 OTU18 OTU19
##     1     2     1     3     3     1     3     1     1     3
## Levels: 1 2 3 4
```

To better visualize the results and the distribution of the clusters, we can plot the histogram of the clusters.

```
library(ggplot2)
library(patchwork) # TO arrange several plots as a grid
plot1 <- ggplot(as.data.frame(rowData(tse)), aes(x =
 ↪  clusters_euclidean)) +
    geom_bar() +
    labs(title = "CAG size distribution (Euclidean distance)",
         x = "Clusters", y = "Feature count (n)")
plot2 <- ggplot(as.data.frame(rowData(tse)), aes(x =
 ↪  clusters_kendall)) +
    geom_bar() +
    labs(title = "CAG size distribution (1 - tau)",
         x = "Clusters", y = "Feature count (n)")
plot1 + plot2 + plot_layout(ncol = 2)
```



It's also possible to merge the rows by cluster.

```
# Aggregate clusters as a sum of each cluster values
tse_merged <- mergeRows(tse, clusters_euclidean)
tse_merged
```

```
## class: TreeSummarizedExperiment
## dim: 5 58
## metadata(0):
## assays(3): counts clr z
## rownames(5): 1 2 3 4 5
## rowData names(8): kingdom phylum ... hclustEuclidean hclustKendall
## 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
```

We can note that it worked as planned since there were 5 clusters and there are now 5 rows.

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 ('transformAssay()') 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.

```
tse <- tseGlobalPatterns
tse <- transformAssay(tse, assay.type = "counts", method =
 ↪  "relabundance", pseudocount = 1)
tse <- transformAssay(x = tse, assay.type = "relabundance",
 ↪  method = "clr",
                      pseudocount = 1, name = "clr")

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

```
##                                         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 <- transformAssay(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 4
## names(4): counts relabundance clr pa
```

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.type = "counts",
                             index = "observed",
                             name="observed")

head(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]))
```

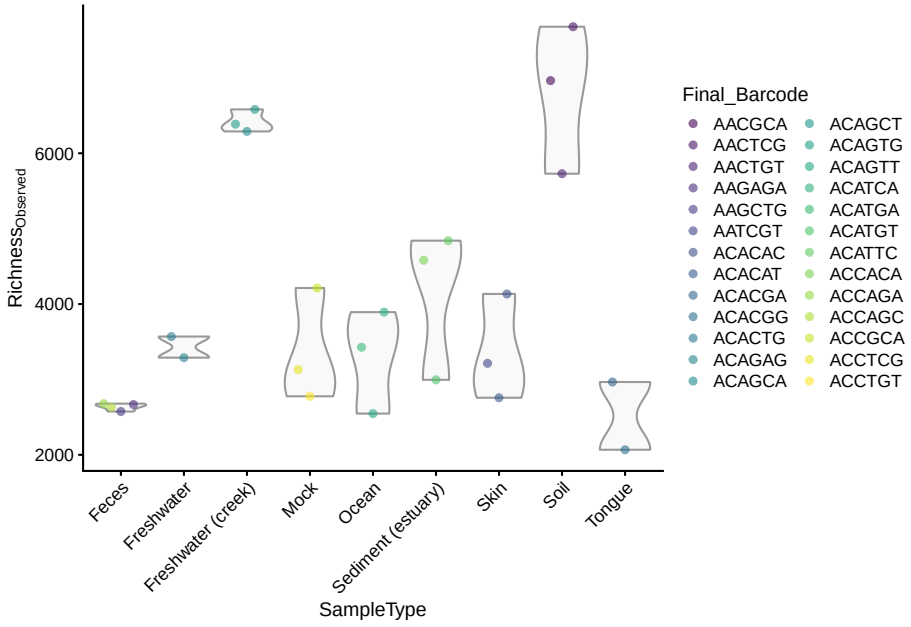


Figure 7.1: Shannon diversity estimates plotted grouped by sample type with colour-labeled barcode.

7.1.2 Diversity

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

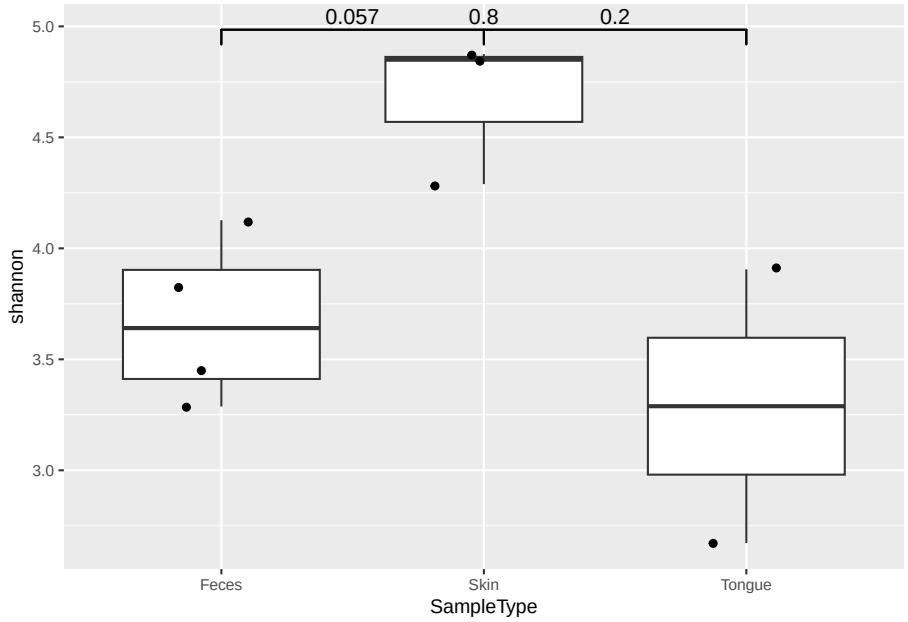
```
tse <- mia::estimateDiversity(tse,
                              assay.type = "counts",
                              index = "shannon",
                              name = "shannon")
head(tse$shannon)
```

```
##     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.

```
library(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)[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`.

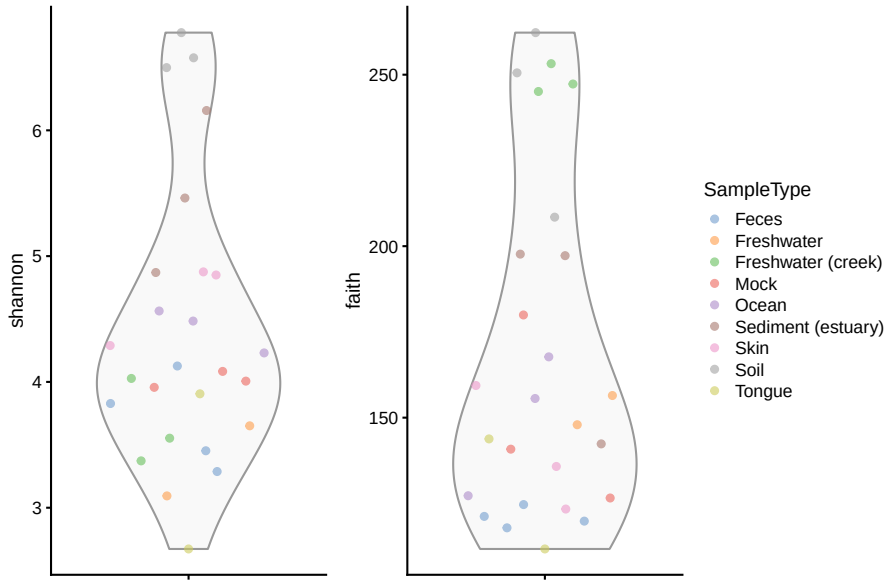
```
tse <- mia::estimateFaith(tse,
                          assay.type = "counts")
head(tse$faith)
```

```
## [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.type = "counts",
                              index = "faith",
                              name = "faith")
```

7.1.4 Evenness

Evenness can be calculated with `estimateEvenness`.

```
tse <- estimateEvenness(tse,
                        assay.type = "counts",
                        index="simpson")
head(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.type = "counts",
                         index="relative")

head(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.type = "counts",
                              index = "log_modulo_skewness")

head(tse$log_modulo_skewness)
```

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

7.1.7 Divergence

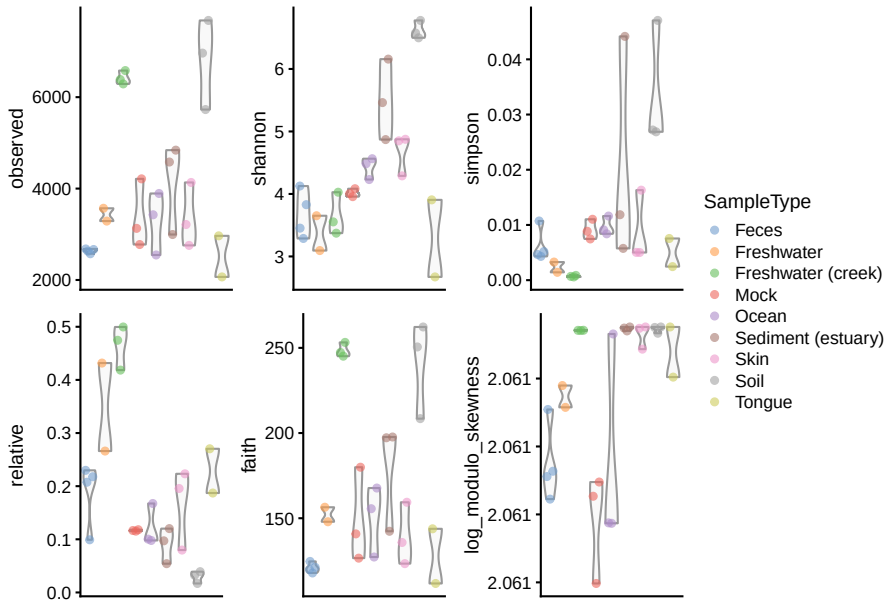
Divergence can be evaluated with `estimateDivergence`. Reference and algorithm for the calculation of divergence can be specified as `reference` and `FUN`, respectively.

```
tse <- mia::estimateDivergence(tse,
                               assay.type = "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")
```



Chapter 8

Community Similarity

Whereas alpha diversity focuses on community variation within a community (one sample), beta diversity quantifies the dissimilarity between communities (multiple samples). In microbiome research, the most popular metrics of beta diversity include the 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 distance (that takes into account the phylogenetic tree information). Notably, only some of these measures are actual *distances*, as this is a mathematical concept whose definition is not satisfied by certain ecological measure, such as the Bray-Curtis index. Therefore, the terms dissimilarity and beta diversity are preferred.

In practice, beta diversity is usually represented as a `dist` object, a triangular matrix where the distance between each pair of samples is encoded by a specific cell. This distance matrix can then undergo ordination, which is an important ecological tool to reduce the dimensionality of data for a more efficient analysis and visualization. Ordination techniques aim to capture as much essential information from the data as possible and turn it into a lower dimensional representation. Dimension reduction is bound to lose information but commonly used ordination techniques can 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?]

Based on the type of algorithm, ordination methods in microbiome research can be generally divided in two categories: unsupervised and supervized ordination. The former includes Principal Coordinate Analysis (PCoA), Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP), whereas the latter is mainly represented by distance-based Redundance Analysis (dbRDA). We will first discuss unsupervized ordination methods and then proceed to supervized ones.

To run the examples in this chapter, the following packages should be imported:

- mia: microbiome analysis framework
- scater: plotting reduced dimensions
- vegan: ecological distances
- ggplot2: plotting
- patchwork: combining plots

8.1 Unsupervized ordination

Unsupervized ordination methods variation in the data without additional information on covariates or other supervision of the model. Among the different approaches, Multi-Dimensional Scaling (MDS) and non-metric MDS (NMDS) can be regarded as the standard. They are jointly referred to as PCoA. For this demonstration we will analyse beta diversity in GlobalPatterns, and observe the variation between stool samples and those with a different origin.

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

# Data matrix (features x samples)
tse <- GlobalPatterns

# some beta diversity metrics are usually applied to relative
 ↪  abundances
tse <- transformAssay(tse,
                      method = "relabundance")

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

8.1.1 Comparing communities by beta diversity analysis

A typical comparison of community compositions starts with a visual representation of the groups by a 2D ordination. Then we estimate relative abundances and MDS ordination based on Bray-Curtis index between the groups, and visualize the results.

In the following examples dissimilarity is calculated with the function supplied to the `FUN` argument. Several metrics of beta diversity are defined by the `vegdist` function of the vegan package, which is often used in this context. However, such custom functions created by the user also work, as long as they return a `dist` object. In either case, this function is then applied to calculate reduced

dimensions via an ordination method, the results of which can be stored in the `reducedDim` slot of the TreeSE. This entire process is contained by the `runMDS` and `runNMDS` functions.

```
# Perform PCoA
tse <- runMDS(tse,
              FUN = vegan::vegdist,
              method = "bray",
              name = "PCoA_BC",
              assay.type = "relabundance")
```

Sample dissimilarity can be visualized on a lower-dimensional display (typically 2D) using the `plotReducedDim` function from the scater package. This also provides tools to incorporate additional information encoded by color, shape, size and other aesthetics. Can you find any difference between the groups?

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

# Calculate explained variance
e <- attr(reducedDim(tse, "PCoA_BC"), "eig")
rel_eig <- e / sum(e[e > 0])

# Add explained variance for each axis
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 = ""))

p
```

With additional tools from the ggplot2 package, ordination methods can be compared to find similarities between them or select the most suitable one to visualize beta diversity in the light of the research question.

```
tse <- runMDS(tse,
              FUN = vegan::vegdist,
              name = "MDS_euclidean",
              method = "euclidean",
              assay.type = "counts")

tse <- runNMDS(tse,
               FUN = vegan::vegdist,
               name = "NMDS_BC")
```

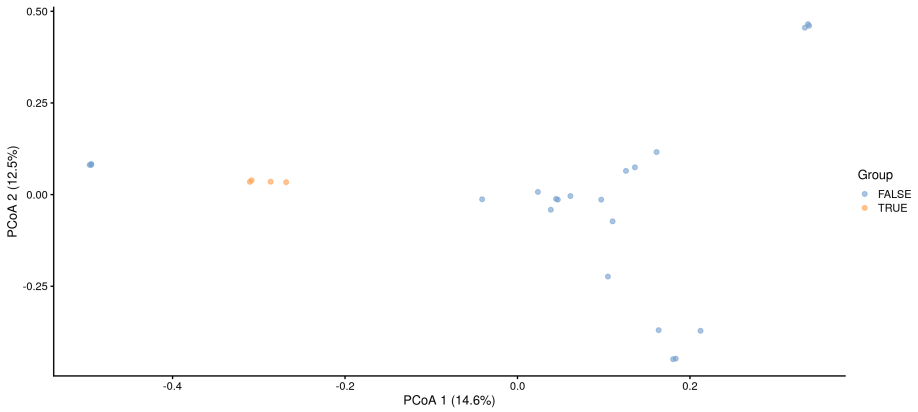


Figure 8.1: MDS plot based on the Bray-Curtis distances on the GlobalPattern dataset.

```
## 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")

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

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`.

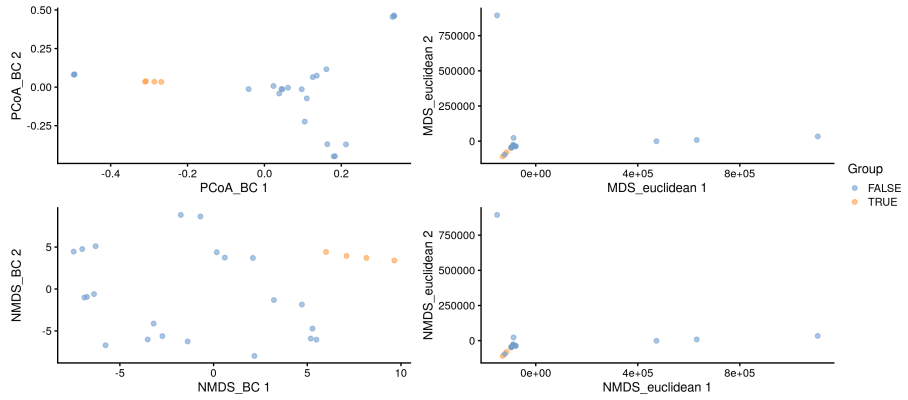


Figure 8.2: Comparison of MDS and NMDS plots based on the Bray-Curtis or euclidean distances on the GlobalPattern dataset.

```
tse <- runMDS(tse,
              FUN = mia::calculateUnifrac,
              name = "Unifrac",
              tree = rowTree(tse),
              ntop = nrow(tse),
              assay.type = "counts")
```

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

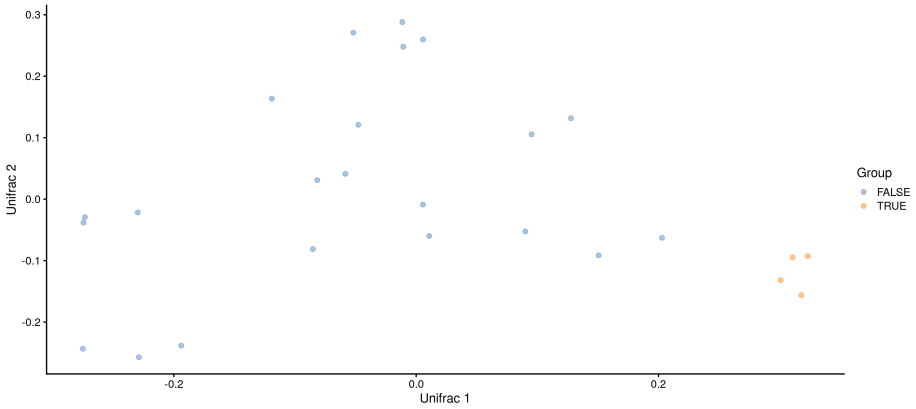


Figure 8.3: Unifrac distances scaled by MDS of the GlobalPattern dataset.

8.1.2 Other ordination methods

Other dimension reduction methods, such as PCA and UMAP, are inherited from the scater package.

```
tse <- runPCA(tse,
              name = "PCA",
              assay.type = "counts",
              ncomponents = 10)

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

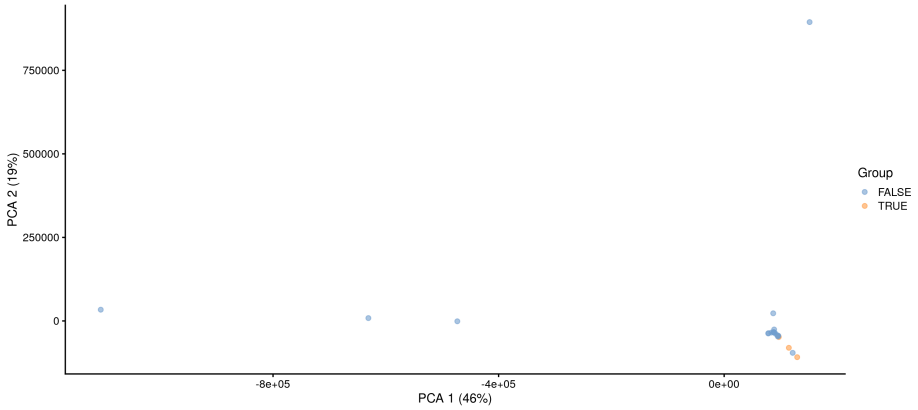


Figure 8.4: PCA plot on the GlobalPatterns data set containing sample from different sources.

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

```
tse <- runUMAP(tse,
               name = "UMAP",
               assay.type = "counts",
               ncomponents = 3)

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

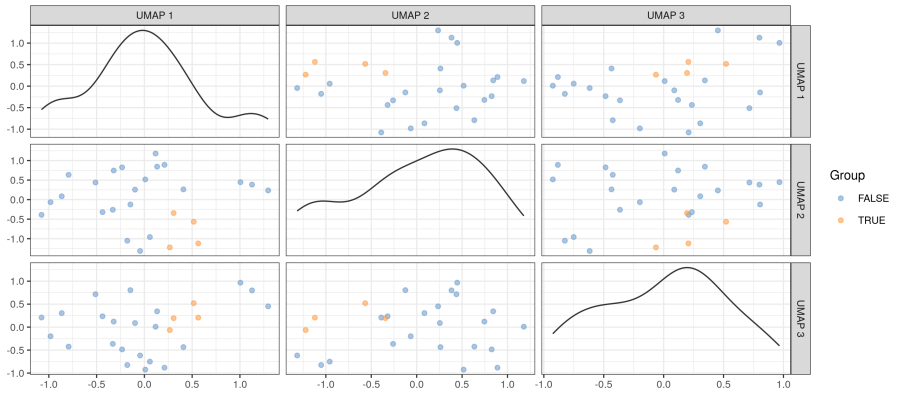


Figure 8.5: UMAP plot on the GlobalPatterns data set containing sample from different sources.

8.1.3 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.

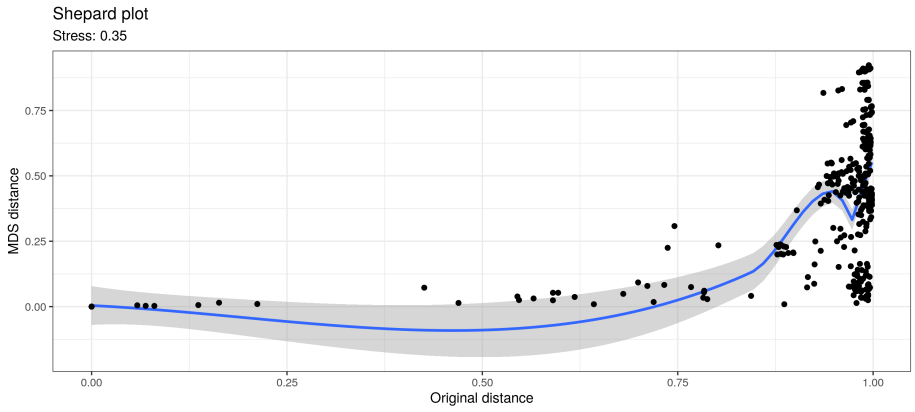
Hence, we propose that for PCoA and other ordination methods, users would report relative stress, which varies in the unit interval and is better if smaller. This can be calculated as shown below.

```
# Quantify dissimilarities in the original feature space
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)
```

A Shepard plot visualizes the original versus the ordinated dissimilarity between the observations.

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

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



8.2 Supervized ordination

dbRDA is a supervized counterpart of PCoA, that is, it takes into account the covariates specified by the user to maximize the variance with respect to the them. The result shows how much each covariate affects beta diversity. The table below illustrates the relation between supervized and unsupervized ordination methods.

	supervized ordination	unsupervized ordination
Euclidean distance	RDA	PCA
non-Euclidean distance	dbRDA	PCoA

We demonstrate the usage of dbRDA with the enterotype TreeSE, where samples correspond to patients. The colData contains the clinical status of each patient as well as a few covariates such as their gender and age, which can be included in the supervized model together with the clinical status, the main outcome variable. dbRDA can be perfomed with the `calculateRDA` function, which lies at the center of the following workflow.

```
# Load packages
library(stringr)

# Load data
data("enterotype", package = "mia")

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

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

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

# # Perform RDA
rda <- calculateRDA(enterotype,
                    assay.type = "relabundance",
                    formula = formula,
                    distance = "bray",
                    na.action = na.exclude)

# Get the rda object
```

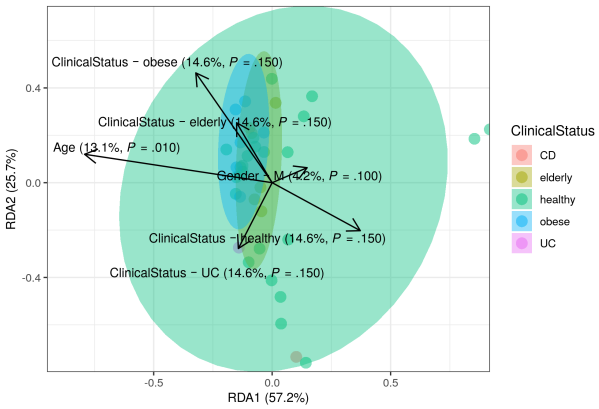
```
rda <- attr(rda, "rda")
# Calculate p-value and variance for whole model
# Recommendation: use 999 permutations instead of 99
set.seed(436)
permanova <- anova.cca(rda, permutations = 99)
# Create a data.frame for results
rda_info <- as.data.frame(permanova)["Model", ]

# 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

knitr::kable(rda_info)
```

	Df	SumOfSqs	F	Pr(>F)	Total variance	Explained variance	Homogeneity p-value
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
library(ggord)

# 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.

8.3 Case studies

8.3.0.1 Visualizing the most dominant genus on PCoA

In this section, we visualize the most dominant genus on PCoA. A similar visualization was proposed by (2021). First, we agglomerate the data at the Genus level and get the dominant taxa per sample.

```
# Agglomerate to genus level
tse_genus <- agglomerateByRank(tse,
                               rank = "Genus")

# Convert to relative abundances
tse_genus <- transformAssay(tse,
                            method = "relabundance",
                            assay.type = "counts")

# Add info on dominant genus per sample
tse_genus <- addPerSampleDominantFeatures(tse_genus,
                                          assay.type =
                                           ↪  "relabundance",
                                          name = "dominant_taxa")
```

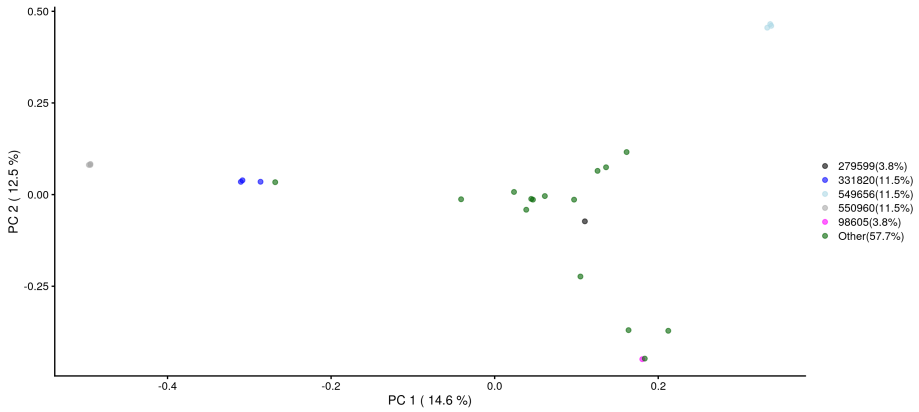
Next, we perform PCoA with Bray-Curtis dissimilarity.

```
tse_genus <- runMDS(tse_genus,
                    FUN = vegan::vegdist,
                    name = "PCoA_BC",
                    assay.type = "relabundance")
```

Finally, we get the top taxa and and visualize their abundances on PCoA. Note that A 3D interactive version of the plot below can be found in 18.

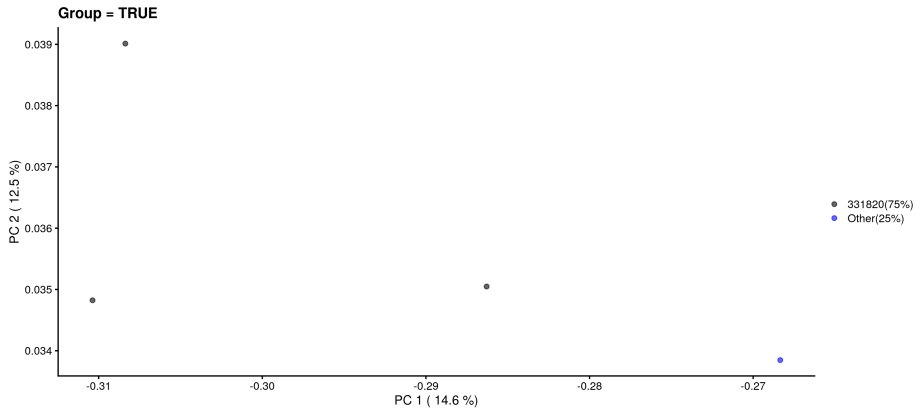
```
# Getting the top taxa
top_taxa <- getTopFeatures(tse_genus,
                       top = 6,
                       assay.type = "relabundance")

# 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

# Define colors for visualization
my_colors <- c("black", "blue", "lightblue", "darkgray",
 ↪  "magenta", "darkgreen", "red")

# Visualization
plot <-plotReducedDim(tse_genus, "PCoA_BC",
                      colour_by = "most_abundant") +
  scale_colour_manual(values = my_colors,
                      labels =
                       ↪  paste0(names(most_abundant_percent),
                       ↪  "(", most_abundant_percent, "%)")) +
  labs(x = paste("PC 1 (", round(var_explained[1], 1), "%)"),
       y = paste("PC 2 (", round(var_explained[2], 1), "%)"),
       color = "")
```

```
plot
```

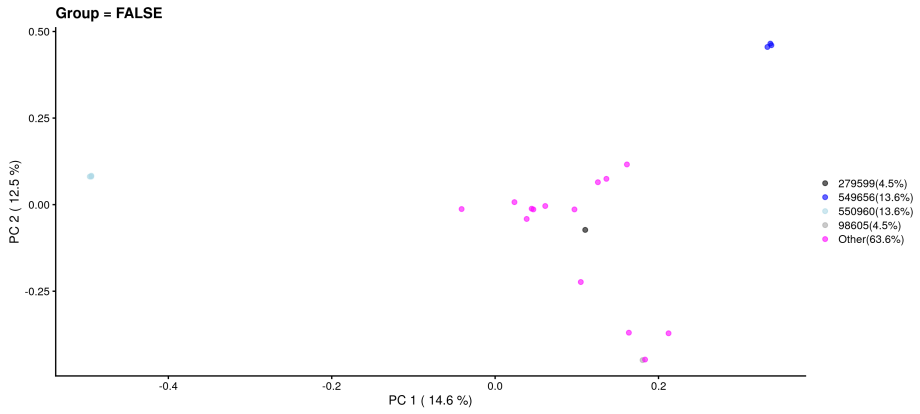


Similarly, we visualize and compare the sub-population.

```
# 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 = my_colors,
                      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 = my_colors,
                      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 = "")
```



8.3.1 Testing differences in community composition between sample groups

Permutational Analysis of Variance (PERMANOVA; (2001)) is a widely used non-parametric multivariate method that aims to estimate the actual statistical

significance of differences in the observed community composition between two groups of samples.

PERMANOVA tests the hypothesis that the centroids and dispersion of the community are equivalent between the compared groups. A p-value smaller than the significance threshold indicates that the groups have a different community composition. This method is implemented with the `adonis2` function from the vegan package.

By default, the argument `by` is set to `"terms"`, in which the order of variables in the formula matters. In this case, each variable is analyzed sequentially, and the result is different when more than 1 variable is introduced and their order differs. Therefore, it is recommended to set `by = "margin"`, which specifies that the marginal effect of each variable is analyzed individually. You can view a comparison between the two designs in chapter 18.1.

We can perform PERMANOVA either with `adonis2` function or by first performing dbRDA and then applying permutational test its results. An advantage of the latter approach is that by doing so we can get coefficients: how much each taxa affects the variation between communities.

```
# 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
```

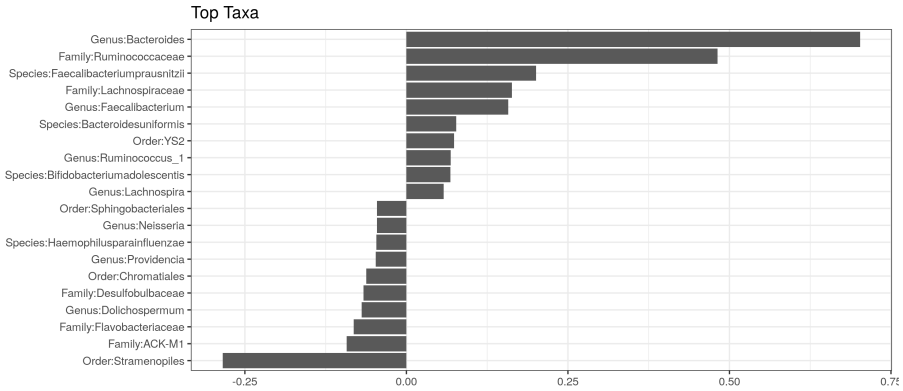
```
##                 p_values
## adonis2             0.02
## dbRDA+anova.cca     0.02
```

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)]
```

```
df <- data.frame(x = top.coef,
                 y = factor(names(top.coef),
                  ↪  unique(names(top.coef))))

ggplot(df, aes(x = x, y = y)) +
  geom_bar(stat = "identity") +
  labs(x = "", y= "", title = "Top Taxa") +
  theme_bw()
```



In the example above, 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.

8.3.2 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.

8.4 Summary

As a final note, we provide a comprehensive list of functions for the evaluation of dissimilarity indices available in the mia and scater packages. The `calculate` methods return a reducedDim object as an output, whereas the `run` methods store the reducedDim object into the specified TreeSE.

- Canonical Correspondence Analysis (CCA): `calculateCCA` and `runCCA`
- dbRDA: `calculateRDA` and `runRDA`
- Double Principal Coordinate Analysis (DPCoA): `calculateDPCoA` and `runDPCoA`
- Jensen-Shannon Divergence (JSD): `calculateJSD` and `runJSD`
- MDS: `calculateMDS` and `runMDS`
- NMDS: `calculateNMDS` and `runNMDS`
- Overlap: `calculateOverlap` and `runOverlap`
- t-distributed Stochastic Neighbor Embedding (t-SNE): `calculateTSNE` and `runTSNE`
- UMAP: `calculateUMAP` and `runUMAP`

For more information on clustering samples by beta diversity, you can refer to:

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

Chapter 9

Community Composition

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

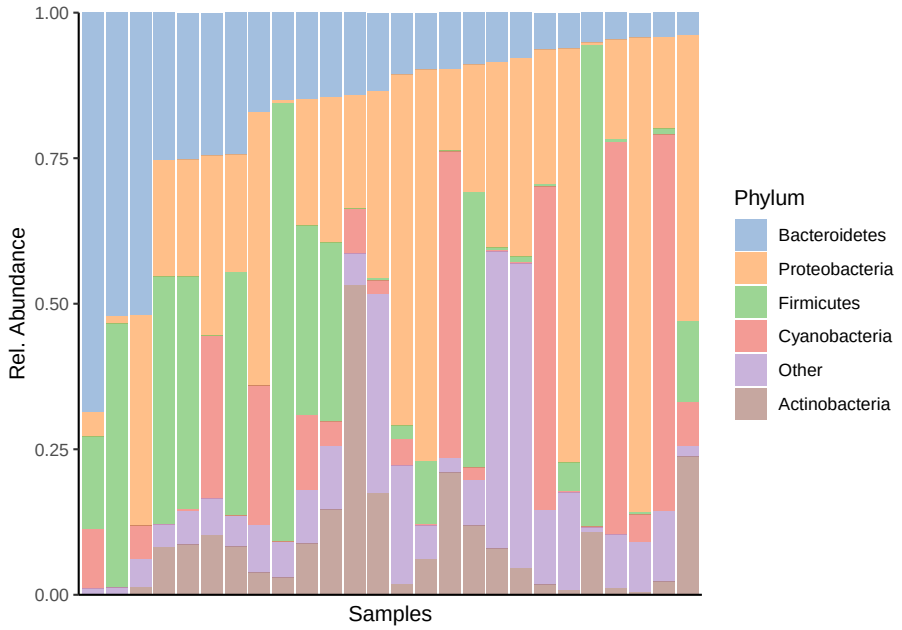
9.1 Visualizing taxonomic composition

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 <- transformAssay(tse, assay.type = "counts", method =
 ↪  "relabundance")

# Getting top taxa on a Phylum level
tse_phylum <- agglomerateByRank(tse, rank ="Phylum",
 ↪  onRankOnly=TRUE)
top_taxa <- getTopFeatures(tse_phylum,top = 5, assay.type =
 ↪  "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.type="relabundance", rank = "Phylum",
              order_rank_by="abund",
              order_sample_by = "Bacteroidetes")
```



9.1.2 Composition heatmap

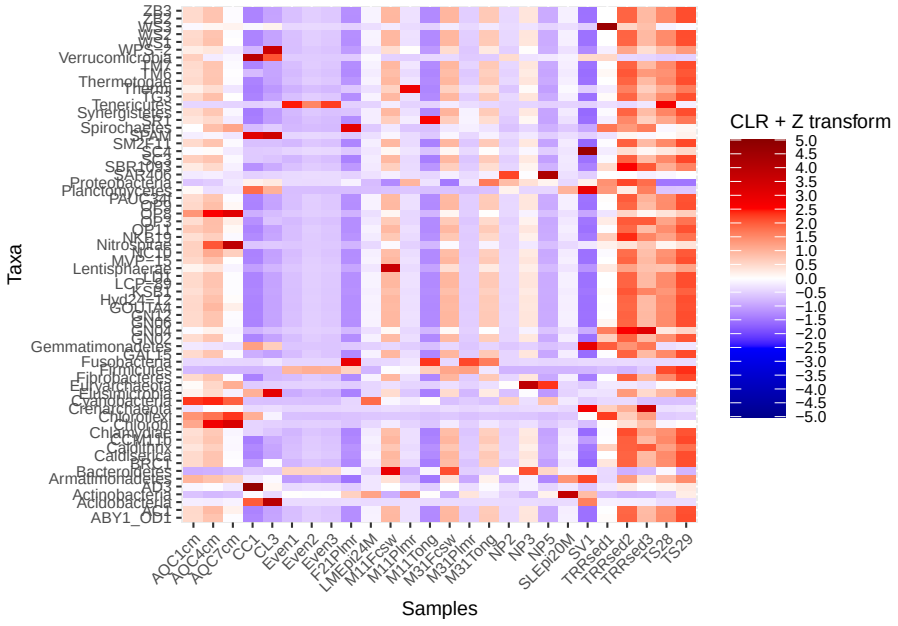
Community composition can be visualized with heatmap, where the horizontal axis represents samples and the vertical axis the taxa. Color of each intersection point represents abundance of a taxon in a specific sample.

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
tse_phylum <- transformAssay(tse_phylum, assay.type = "counts",
                             method = "relabundance",
                              ↪  pseudocount = 1)

tse_phylum <- transformAssay(tse_phylum, assay.type =
 ↪  "relabundance",
                             method = "clr", pseudocount = 1)

# Add z-transformation on features (taxa)
tse_phylum <- transformAssay(tse_phylum, assay.type = "clr",
                             MARGIN = "features",
                             method = "z", name = "clr_z")
```

Visualize as heatmap.

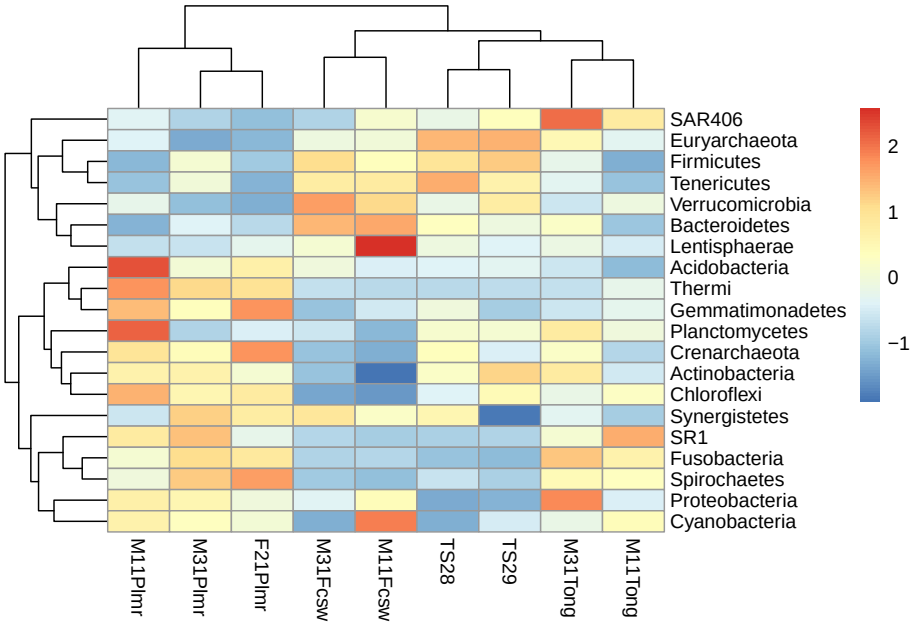
```
# Melt the assay for plotting purposes
df <- meltAssay(tse_phylum, assay.type = "clr_z")

# 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.

```
library(pheatmap)

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

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

tse_phylum_subset <- transformAssay(tse_phylum_subset, assay.type
 ↪  = "clr",
                                                MARGIN = "features",
                                                method = "z", name =
                                                 ↪  "clr_z")

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

# Gets the assay table
mat <- assay(tse_phylum_subset, "clr_z")
```

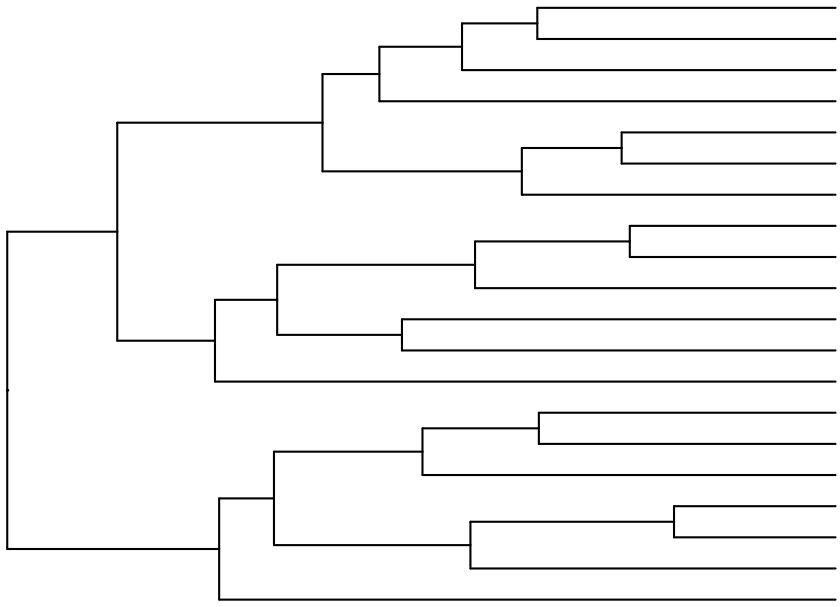
```
# Creates the heatmap
pheatmap(mat)
```



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

```
library(ape)

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

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

```
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
```

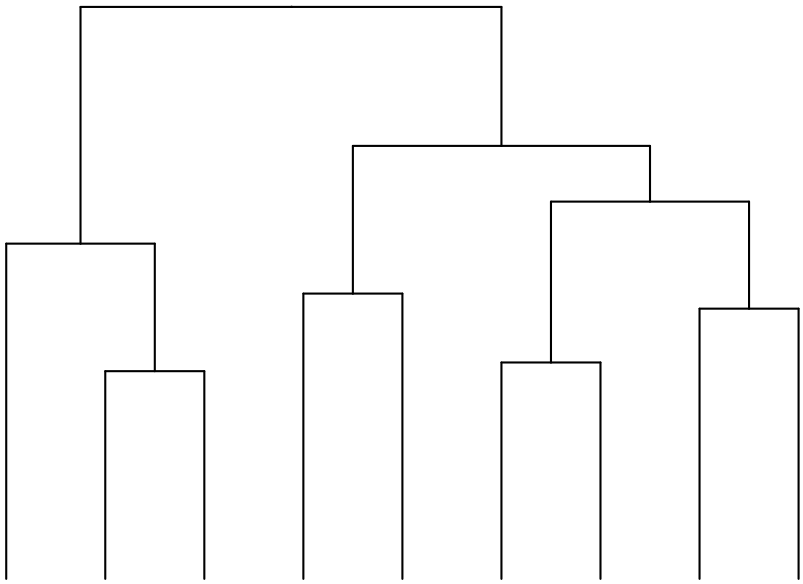
```
##                  clusters
## Chloroflexi             3
## Actinobacteria          3
## Crenarchaeota           3
## Planctomycetes          3
## Gemmatimonadetes        3
## Thermi                  3
```

```
## 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
```



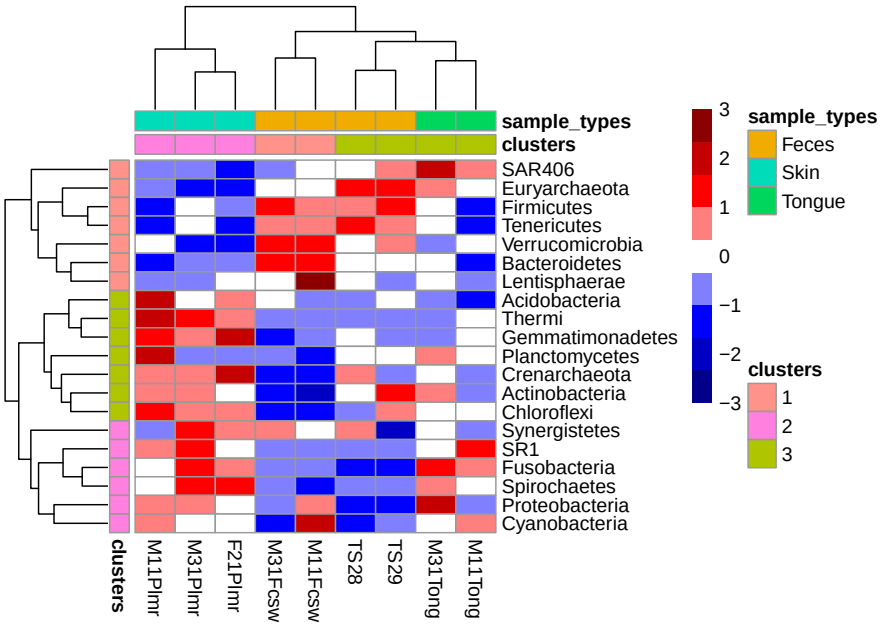
```
# Creates clusters
sample_clusters <- factor(cutree(tree = sample_hclust, k = 3))

# 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.

Chapter 10

Community Typing (Clustering)

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

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

As mentioned before, clustering can be done either features or samples. We will focus on the latter here. To learn about feature clustering, check out chapter 6.3.

10.1 Custom tools

bluster is a Bioconductor package providing tools for clustering data in in the `SummarizedExperiment` container. It offers multiple algorithms such as hierarchical clustering, DBSCAN, K-means, amongst others. The first thing to do when using this package is to load it, and transform the data if necessary, depending on your analysis goals.

```
# Load dependencies
library(bluster)
```

```
# Apply transformation
tse <- transformAssay(tse, method = "relabundance")
```

The main focus here will be how to use mia's `cluster` function to cluster. It has multiple parameters that allow you to shape the result. * The main new parameter allows you to choose the algorithm you want to use. In this example, we will use HclustParam which does the hierarchical clustering. This parameter itself has parameters on its own, you can check them in the HclustParam documentation. * Another parameter is `MARGIN`, which allows us to choose whether we want to cluster the features or samples . Here we will cluster the latter. * We will see the other parameters as we go along.

```
# Simple use of the hierarchical clustering. Here, the default
 ↪  parameters
# set the cut height to half of the dendrogram height.
tse <- cluster(tse, assay.type = "relabundance",
               MARGIN = "samples", HclustParam())

# Check the result contained in the clusters part of colData
colData(tse)$clusters
```

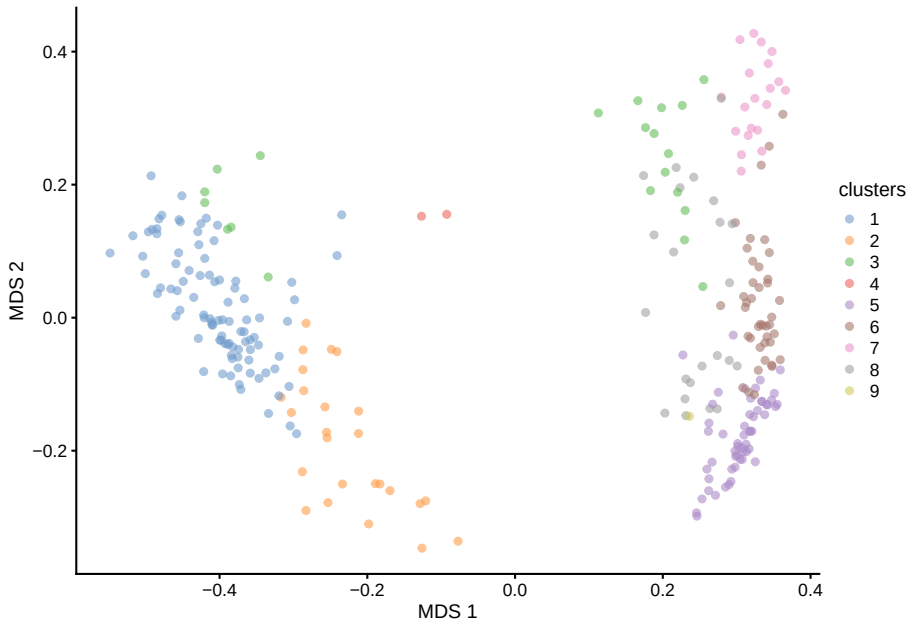
```
##    AM.AD.1    AM.AD.2  AM.F10.T1  AM.F10.T2    DA.AD.1   DA.AD.1T    DA.AD.2
##          1          1          1          1          1          1          1
##    DA.AD.3   DA.AD.3T    DA.AD.4    ES.AD.1    ES.AD.2    ES.AD.3    ES.AD.4
##          1          1          1          2          3          2          1
##    FR.AD.1    FR.AD.2    FR.AD.3    FR.AD.4    FR.AD.5    FR.AD.6    FR.AD.7
##          1          1          2          1          1          1          1
##    FR.AD.8    IT.AD.1    IT.AD.2    IT.AD.3    IT.AD.4    IT.AD.5    IT.AD.6
##          2          2          1          2          3          1          1
##    JP.AD.1    JP.AD.2    JP.AD.3    JP.AD.4    JP.AD.5    JP.AD.6    JP.AD.7
##          2          1          1          2          1          2          2
##    JP.AD.8    JP.AD.9    JP.IN.1    JP.IN.2    JP.IN.3    JP.IN.4     MH0001
##          2          2          1          4          4          1          3
##     MH0002     MH0003     MH0004     MH0005     MH0006     MH0007     MH0008
##          1          1          1          1          1          1          2
##     MH0009     MH0010     MH0011     MH0012     MH0013     MH0014     MH0015
##          1          2          1          1          1          1          1
##     MH0016     MH0017     MH0018     MH0019     MH0020     MH0021     MH0022
##          2          1          3          2          2          1          1
##     MH0023     MH0024     MH0025     MH0026     MH0027     MH0028     MH0030
##          1          1          1          1          2          1          1
##     MH0031     MH0032     MH0033     MH0034     MH0035     MH0036     MH0037
##          1          3          1          3          1          1          1
```

```
##     MH0038     MH0039     MH0040     MH0041     MH0042     MH0043     MH0044
##          1          1          1          2          1          1          1
##     MH0045     MH0046     MH0047     MH0048     MH0049     MH0050     MH0051
##          2          1          1          1          1          1          1
##     MH0052     MH0053     MH0054     MH0055     MH0056     MH0057     MH0058
##          1          1          1          1          1          1          1
##     MH0059     MH0060     MH0061     MH0062     MH0063     MH0064     MH0065
##          1          1          2          1          1          1          1
##     MH0066     MH0067     MH0068     MH0069     MH0070     MH0071     MH0072
##          1          1          2          1          1          1          2
##     MH0073     MH0074     MH0075     MH0076     MH0077     MH0078     MH0079
##          2          1          1          1          1          1          1
##     MH0080     MH0081     MH0082     MH0083     MH0084     MH0085     MH0086
##          1          1          1          1          3          2          1
##     TS1_V2    TS10_V2   TS100_V2 TS101.2_V2   TS103_V2   TS104_V2   TS105_V2
##          5          6          5          7          8          5          7
##    TS106_V2   TS107_V2   TS109_V2    TS11_V2   TS110_V2   TS111_V2   TS115_V2
##          6          8          5          5          7          6          3
##    TS116_V2   TS117_V2   TS118_V2   TS119_V2    TS12_V2   TS120_V2   TS124_V2
##          8          6          5          6          5          3          6
##    TS125_V2   TS126_V2   TS127_V2   TS128_V2   TS129_V2    TS13_V2   TS130_V2
##          6          8          6          5          5          5          5
##    TS131_V2   TS132_V2   TS133_V2   TS134_V2   TS135_V2   TS136_V2   TS137_V2
##          6          5          5          6          5          7          5
##    TS138_V2   TS139_V2    TS14_V2   TS140_V2   TS141_V2   TS142_V2   TS143_V2
##          7          7          7          3          5          6          6
##    TS144_V2   TS145_V2   TS146_V2   TS147_V2   TS148_V2   TS149_V2    TS15_V2
##          3          5          8          6          5          5          3
##    TS150_V2   TS151_V2   TS152_V2   TS153_V2 TS154.2_V2   TS155_V2   TS156_V2
##          7          6          6          6          8          8          5
##     TS16_V2   TS160_V2   TS161_V2   TS162_V2   TS163_V2   TS164_V2   TS165_V2
##          5          3          6          5          5          5          8
##    TS166_V2   TS167_V2   TS168_V2   TS169_V2    TS17_V2   TS170_V2   TS178_V2
##          5          8          6          8          5          7          8
##    TS179_V2   TS180_V2   TS181_V2   TS182_V2   TS183_V2   TS184_V2   TS185_V2
##          7          5          5          5          5          5          6
##    TS186_V2    TS19_V2   TS190_V2   TS191_V2   TS192_V2   TS193_V2   TS194_V2
##          5          5          6          5          6          6          6
##    TS195_V2     TS2_V2    TS20_V2    TS21_V2    TS22_V2    TS23_V2    TS25_V2
##          5          8          3          6          7          7          6
##     TS26_V2    TS27_V2    TS28_V2    TS29_V2     TS3_V2    TS30_V2    TS31_V2
##          5          8          6          7          6          7          6
##     TS32_V2    TS33_V2    TS34_V2    TS35_V2    TS37_V2    TS38_V2    TS39_V2
##          3          7          6          3          5          7          5
##      TS4_V2    TS43_V2    TS44_V2    TS49_V2     TS5_V2    TS50_V2    TS51_V2
##          5          8          8          7          7          7          8
```

```
##    TS55_V2    TS56_V2    TS57_V2     TS6_V2    TS61_V2    TS62_V2    TS63_V2
##          6          5          9          3          6          5          6
##    TS64_V2    TS65_V2    TS66_V2    TS67_V2    TS68_V2    TS69_V2     TS7_V2
##          8          8          8          6          5          8          3
##    TS70_V2    TS71_V2    TS72_V2    TS73_V2    TS74_V2    TS75_V2    TS76_V2
##          8          6          6          5          8          6          5
##    TS77_V2    TS78_V2     TS8_V2    TS82_V2    TS83_V2    TS84_V2    TS85_V2
##          6          3          5          3          6          3          6
##    TS86_V2    TS87_V2    TS88_V2    TS89_V2     TS9_V2    TS90_V2    TS91_V2
##          5          8          5          6          5          5          7
##    TS92_V2    TS94_V2    TS95_V2    TS96_V2    TS97_V2    TS98_V2  TS99.2_V2
##          6          5          5          5          5          5          5
## Levels: 1 2 3 4 5 6 7 8 9
```

Once the clustering on the samples is done, we can also plot the clusters.

```
library(scater)

# Add the MDS dimensions for plotting
tse <- runMDS(tse, assay.type = "relabundance",
              FUN = vegan::vegdist, method = "bray")

# Plot the clusters
plotReducedDim(tse, "MDS", colour_by = "clusters")
```



We will now see different common algorithms and how to use them.

10.2 Hierarchical clustering

The hierarchical clustering algorithm aims to find hierarchy between samples/features. There are to approaches: agglomerative ("bottom-up") and divisive ("top-down").

In agglomerative approach, each observation is first in a unique cluster. The algorithm continues by agglomerating similar clusters. The divisive approach starts with one cluster that contains all the observations. Clusters are split recursively to clusters that differ the most. The clustering ends when each cluster contains only one observation. In this algorithm, the similarity of two clusters is based on the distance between them.

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

```
library(vegan)

# Load experimental data
tse <- enterotype
```

Hierarchical clustering requires 2 steps. 1. Computation of the dissimilarities with a given distance.
2. Clustering based on dissimilarities.

Additionally, since sequencing data is compositional, we'll apply relative transformation (as seen in the previous example).

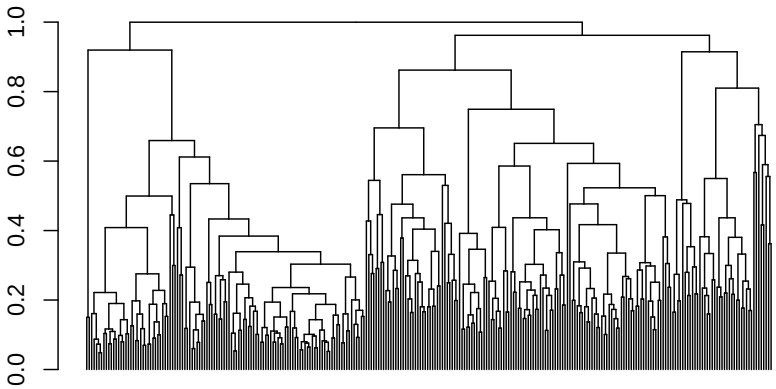
```
# Apply transformation
tse <- transformAssay(tse, method = "relabundance")

# Do the clustering
tse <- cluster(tse,
               assay.type = "relabundance",
               MARGIN = "samples",
               HclustParam(method = "complete",
                           dist.fun = vegdist,
                           metric = "bray"),
               full = TRUE,
               clust.col = "Hclust")
```

In this example, we wanted additional information on the clustering. To do so, we used the `full` parameter. We also computed the dissimilarities with the

bray distance. Finally, the `clust.col` parameter allows us to choose the name of the column in the colData (default name is `clusters`).

Next, we will plot the dendrogram, which is possible since we got the additional information from the clustering.

```
library(dendextend)

# Get hclust data from metadata
hclust_data <- metadata(tse)$clusters$hclust

# Get the dendrogram object
dendro <- as.dendrogram(hclust_data)

# Plot dendrogram

dendro %>% set("labels", NULL) %>% plot()
```



In our case, we cut the dendrogram in half by default. To know how many clusters we have, we can check the colData.

```
# Get the clusters
head(colData(tse)$Hclust)
```

```
##   AM.AD.1   AM.AD.2 AM.F10.T1 AM.F10.T2   DA.AD.1  DA.AD.1T
```

```
##         1         1         2         1         3         3
## 26 Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ... 26
```

We can see that there are 26 clusters, but that probably isn't optimal since the the number of clusters was chosen arbitrarily. To determine the number of clusters, we can use the dendrogram. Usually the tree is split where the branch length is the largest. However, as we can see from the dendrogram, clusters are not clear. There are algorithms to identify the optimal number of clusters.

The `NbClust` library is useful to that end as it offers multiple methods to determine the optimal number of clusters. Here we will use the silhouette analysis to determine the optimal number of clusters. For each data point, this analysis measures the distance to other data points in the same cluster (cohesion), and the distance to the other clusters (separation), establishing a score. That score is then combined across the data points. `NbClust` does this for multiple number of clusters and the best score corresponds to the optimal number of clusters.

```
library(NbClust)
diss <- metadata(tse)$clusters$dist

# Apply the silhouette analysis on the distance matrix
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,
```

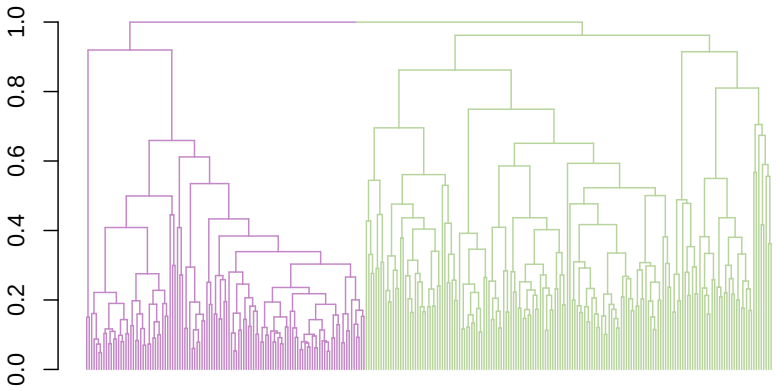
```
res$Best.nc
```

```
## Number_clusters     Value_Index
##          2.0000          0.4783
```

Based on the result, let's divide observations into 2 clusters.

```
library(dendextend)

# Get optimal number of clusters
k <- res$Best.nc[1]

# Making colors for 2 clusters
col_val_map <- randomcoloR::distinctColorPalette(k) %>%
    as.list() %>%
    setNames(paste0("clust_", seq(k)))
```

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



10.3 K-means clustering

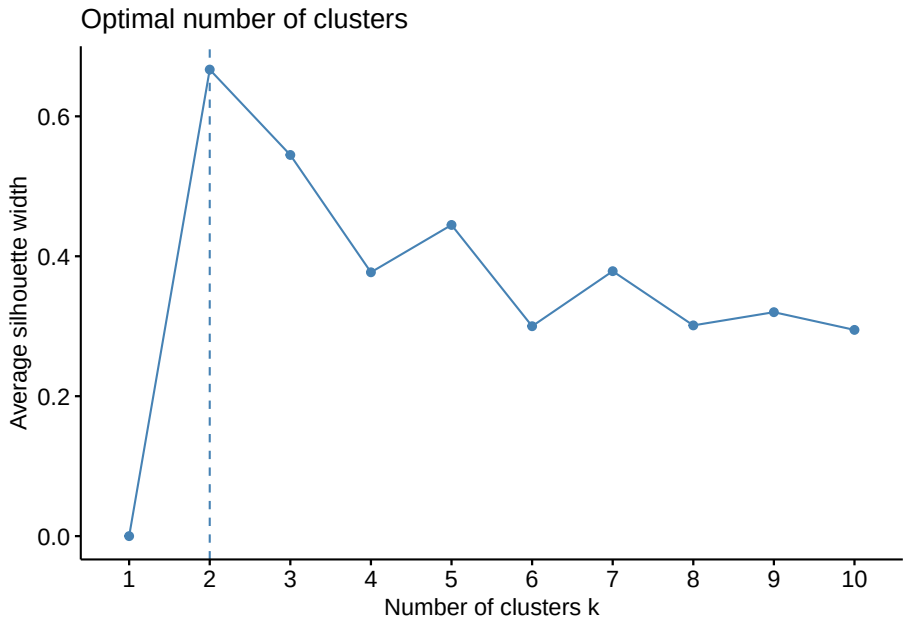
Let's now try k-means clustering. Here observations are divided into clusters so that the distances between observations and cluster centers are minimized; an observation belongs to cluster whose center is the nearest.

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

As an alternative to `NbClust` get the optimal number of clusters, we can visualize the silhouette analysis thanks to the library `factoextra`.

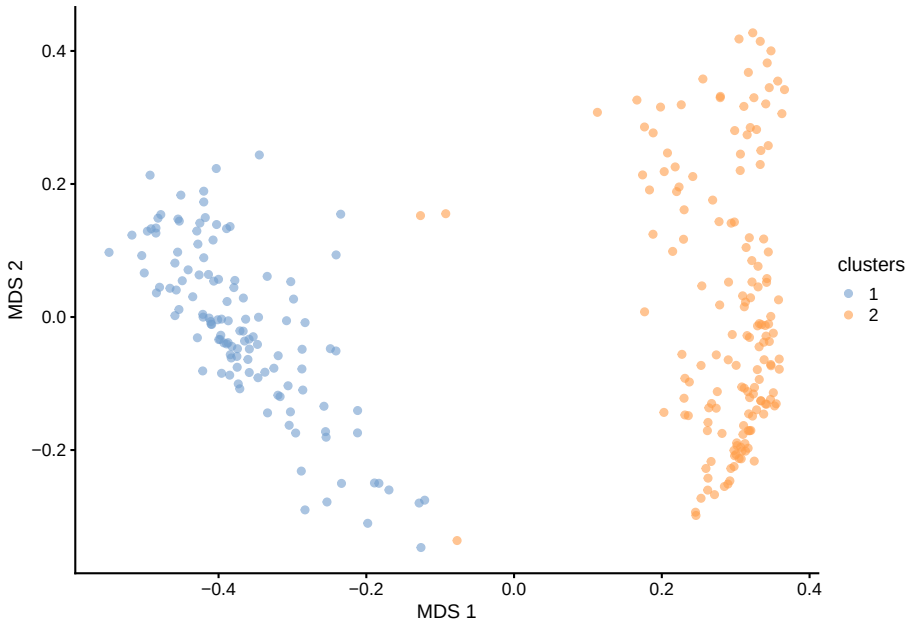
```
library(factoextra)

# 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 confirm that 2 is the optimal number of clusters in k-means clustering.

```
# The first step is random, add seed for reproducibility
set.seed(15463)

# Perform k-means clustering with 2 clusters
km <- kmeans(diss, 2, 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, assay.type = "relabundance",
              FUN = vegan::vegdist, method = "bray")

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



10.4 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. Since the dataset is large, the algorithm will take long computational time. Therefore, we use only a subset of the data; agglomerated by Phylum as a rank.

```
# Get the data
data("GlobalPatterns", package = "mia")
tse <- GlobalPatterns

# Agglomerate by rank
tse <- agglomerateByRank(tse, rank = "Phylum", agglomerateTree =
 ↪  TRUE)
```

Here we will further our use of `cluster` by renaming the clusters column in the metadata thanks to the `name` parameter.

```
# Run the model and calculates the most likely number of clusters
 ↪  from 1 to 7
tse_dmm <- cluster(tse, name = "DMM", DmmParam(k = 1:7, type =
 ↪  "laplace"),
                   MARGIN = "samples", full = TRUE)
```

```
# The dmm info is stored in the metadata under the 'DMM' column
tse_dmm
```

```
## class: TreeSummarizedExperiment
## dim: 67 26
## metadata(2): agglomerated_by_rank DMM
## 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(8): X.SampleID Primer ... Description clusters
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (67 rows)
## rowTree: 1 phylo tree(s) (66 leaves)
## colLinks: NULL
## colTree: NULL
```

The following operation returns a list of DMM objects for closer investigation.

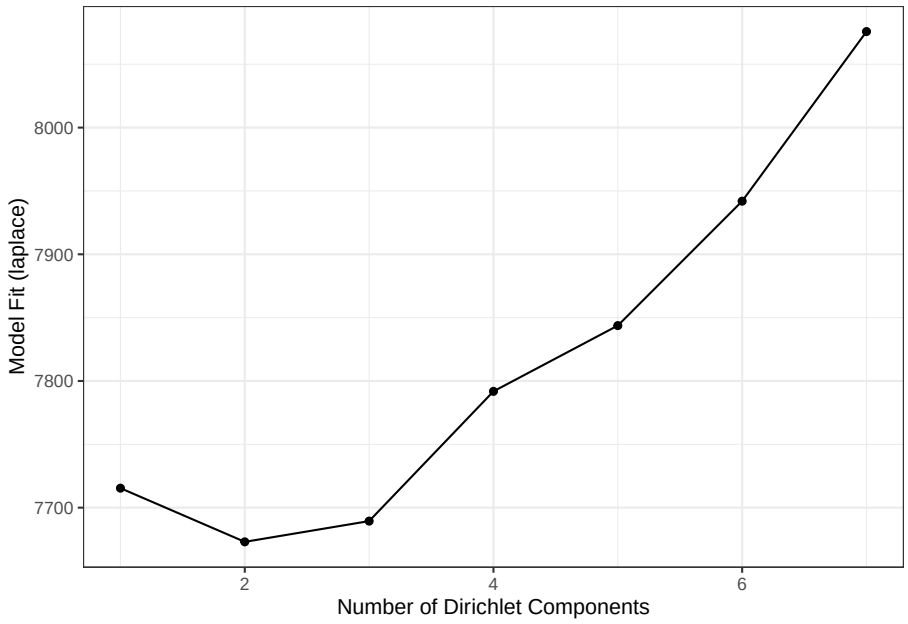
```
metadata(tse_dmm)$DMM$dmm
```

```
## [[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: 7689 BIC: 8076 AIC: 7948
##
## [[4]]
## class: DMN
## k: 4
## samples x taxa: 26 x 67
## Laplace: 7792 BIC: 8357 AIC: 8187
##
## [[5]]
## class: DMN
## k: 5
## samples x taxa: 26 x 67
## Laplace: 7844 BIC: 8548 AIC: 8335
##
## [[6]]
## class: DMN
## k: 6
## samples x taxa: 26 x 67
## Laplace: 7942 BIC: 8822 AIC: 8566
##
## [[7]]
## class: DMN
## k: 7
## samples x taxa: 26 x 67
## Laplace: 8076 BIC: 9100 AIC: 8801
```

We can see the Laplace approximation (model evidence) for each model of the k models.

```
library(miaViz)
plotDMNFit(tse_dmm, type = "laplace", name = "DMM")
```



On the graph, we can see that the best number of clusters is 2. We can confirm that with the following operation.

```
# Get the model that has the best fit
bestFit <-
 ↪  metadata(tse_dmm)$DMM$dmm[[metadata(tse_dmm)$DMM$best]]
bestFit
```

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

10.4.1 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.

```
dmm_group <- calculateDMNgroup(tse_dmm, variable = "SampleType",
                               assay.type = "counts", k = 2,
                               seed = .Machine$integer.max)

dmm_group
```

```
## class: DMNGroup
## summary:
##                     k samples taxa    NLE  LogDet Laplace    BIC  AIC
## Feces               2       4   67 1078.3 -106.19   901.1 1171.9 1213
## Freshwater          2       2   67  889.6  -97.28   716.9  936.4 1025
## Freshwater (creek) 2       3   67 1600.3  860.08  1906.3 1674.5 1735
## Mock                2       3   67 1008.4  -55.37   856.6 1082.5 1143
## Ocean               2       3   67 1096.7  -56.21   944.6 1170.9 1232
## Sediment (estuary) 2       3   67 1195.5   18.63  1080.8 1269.7 1331
## Skin                2       3   67  992.6  -84.81   826.2 1066.8 1128
## Soil                2       3   67 1380.3   11.21  1261.8 1454.5 1515
## Tongue              2       2   67  783.0 -107.74   605.1  829.8  918
```

Mixture weights (rough measure of the cluster size).

```
DirichletMultinomial::mixturewt(bestFit)
```

```
##       pi theta
## 1 0.5385 20.59
## 2 0.4615 15.32
```

It's also possible to get the samples-cluster assignment probabilities: how probable it is that each sample belongs to each cluster

```
prob <- metadata(tse_dmm)$DMM$prob
head(prob)
```

```
##                 1         2
## CL3     1.000e+00 4.594e-17
## CC1     1.000e+00 3.526e-22
## SV1     1.000e+00 1.735e-12
## M31Fcsw 7.385e-26 1.000e+00
## M11Fcsw 1.089e-16 1.000e+00
## M31Plmr 1.150e-13 1.000e+00
```

We can also know the contribution of each taxa to each component

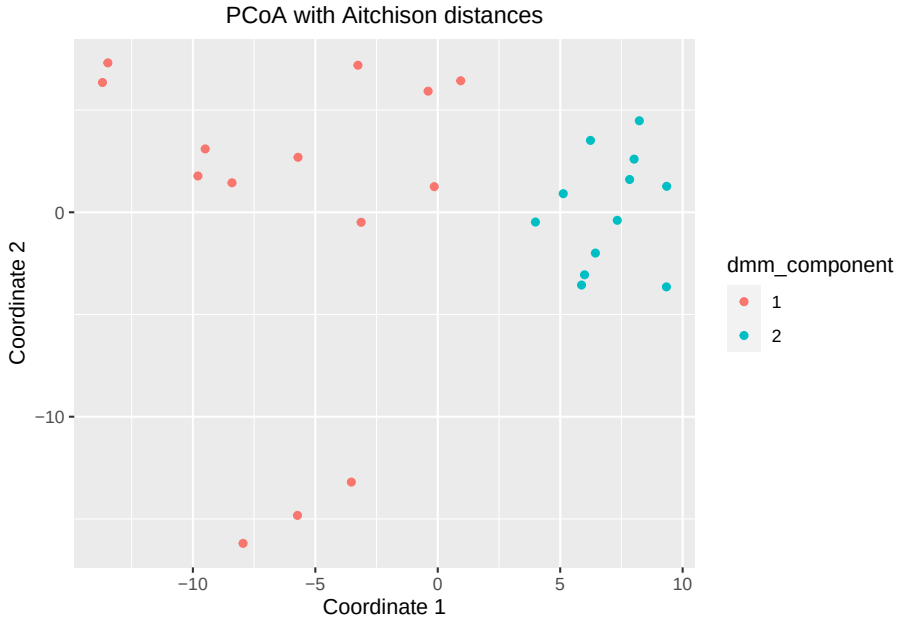
```
head(DirichletMultinomial::fitted(bestFit))
```

```
##                            [,1]      [,2]
## Phylum:Crenarchaeota  0.30381 0.1354058
## Phylum:Euryarchaeota  0.23114 0.1468923
## Phylum:Actinobacteria 1.21375 1.0581803
## Phylum:Spirochaetes   0.21393 0.1318079
## Phylum:MVP-15         0.02983 0.0007714
## Phylum:Proteobacteria 6.84670 1.8116041
```

Finally, to be able to visualize our data and clusters, we start by computing the euclidean PCoA and storing it as a data frame.

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

# Do principal coordinate analysis
df <- calculateMDS(tse, assay.type = "clr", method = "euclidean")

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

```
# Create a data frame that contains principal coordinates and DMM
 ↪  information
euclidean_dmm_pcoa_df <- cbind(euclidean_pcoa_df,
                               dmm_component =
                                ↪  colData(tse_dmm)$clusters)

# Create 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(plot.title = element_text(size = 12, # makes titles
     ↪  smaller
                                    hjust = 0.5))

euclidean_dmm_plot
```



10.5 Graph-based clustering

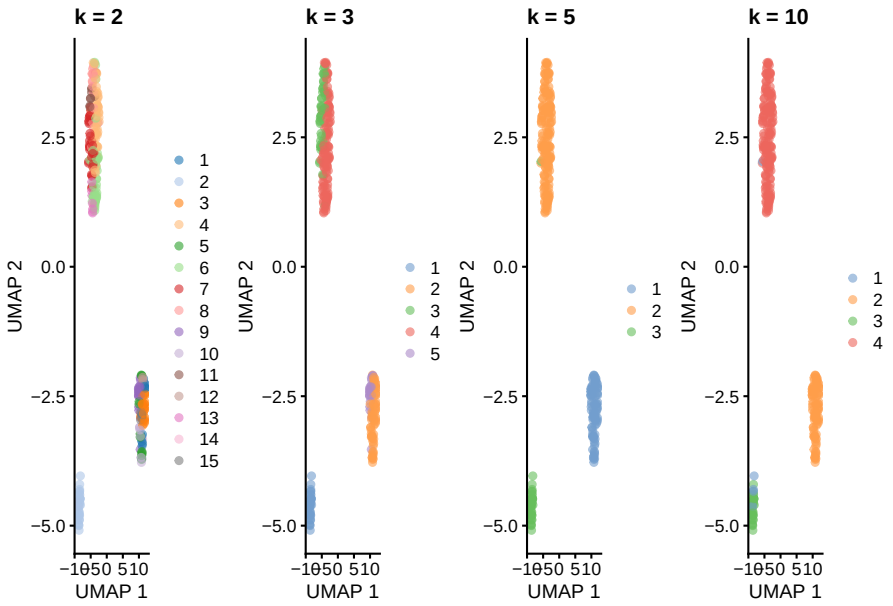
Another approach for discovering communities within the samples of the data, is to run community detection algorithms after building a graph. The following demonstration builds a graph based on the k nearest-neighbors and performs the community detection on the fly.

Here, we will be using the `cluster` function with a graph-based clustering function, enabling the community detection task.

The algorithm used is "short random walks" (Pons and Latapy, 2006). The 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. Let us cluster the `enterotype` dataset.

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

# Get enterotype dataset and transform data with rclr
tse <- enterotype
tse <- transformAssay(tse, method = "rclr")

# Perform and store UMAP
tse <- runUMAP(tse, name = "UMAP", assay.type = "rclr")
```

```
# Set different k values to test
k <- c(2, 3, 5, 10)

ClustAndPlot <- function(x) {
    # Add the clustering data from the short random walks
     ↪ algorithm to the TSE
    tse <- cluster(tse, assay.type = "rclr",
                   MARGIN = "col", NNGraphParam(k = x))

    # Plot the results of the clustering as a color for each
     ↪ sample
    plotUMAP(tse, colour_by = I(colData(tse)$clusters)) +
    labs(title = paste0("k = ", x))
}

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

# Display plots in a grid
plots[[1]] + plots[[2]] + plots[[3]] + plots[[4]] +
 ↪ plot_layout(ncol = 4)
```

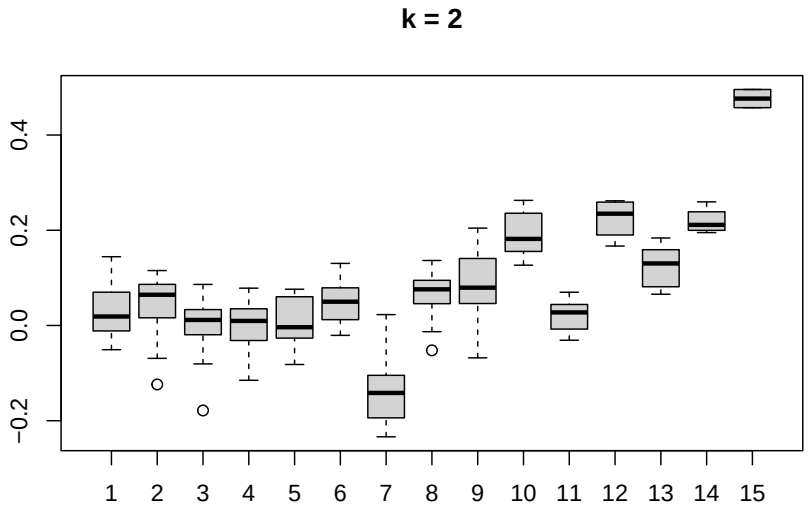


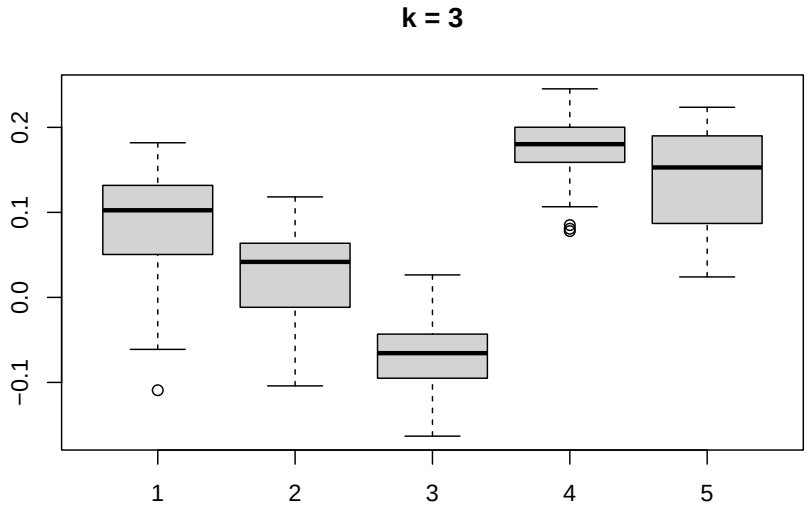
In this graph, we can clearly see the impact of the k choice on the quality of the

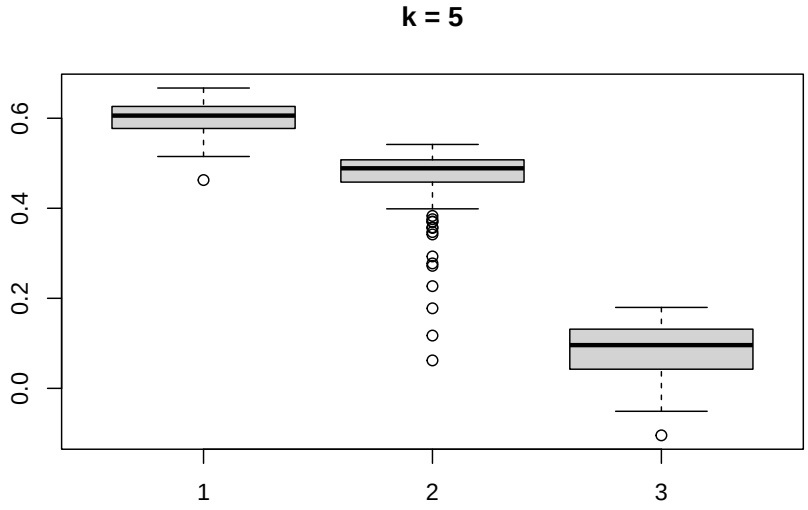
clustering

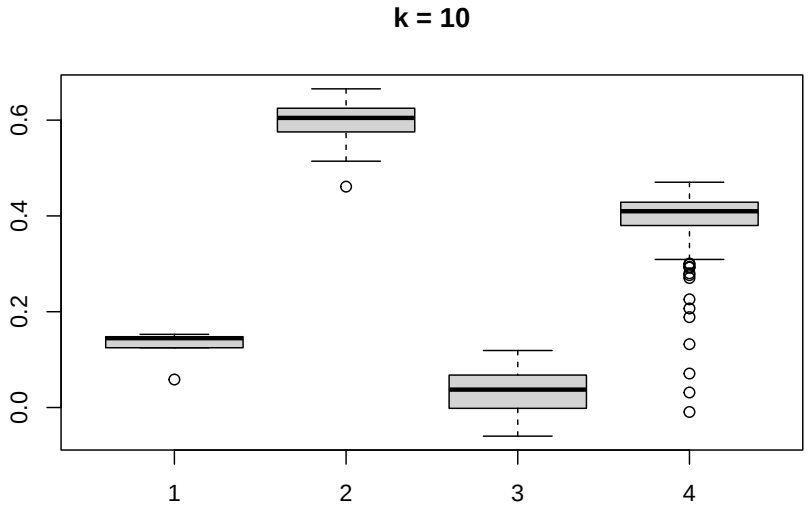
Similarly, the ***bluster*** (Lun, 2021) package offers clustering diagnostics that can be used for judging the clustering quality (see Assorted clustering diagnostics). In the following, Silhouette width as a diagnostic tool is computed and results are visualized for each case presented earlier. For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

```
ClustDiagPlot <- function(x) {
  # Get the clustering results
    tse <- cluster(tse, assay.type = "rclr",
                   MARGIN = "col", NNGraphParam(k = x))

    # Compute the diagnostic info
    sil <- approxSilhouette(t(assays(tse)$rclr),
 ↪  colData(tse)$clusters)

    # Plot as a boxplot to observe cluster separation
    boxplot(split(sil$width, colData(tse)$clusters), main =
     ↪  paste0("k = ", x))
}
# Apply the function for different k values
res <- lapply(k, ClustDiagPlot)
```




k = 3
0.2
0.1
0.0
−0.1
1
2
3
4
5


k = 5
0.6
0.4
0.2
0.0
1
2
3



10.6 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 two different cases and example solutions to apply biclustering to them.

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

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 9.

10.6.1 Taxa vs samples

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

Load example data

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

Only the most prevalent taxa are included in analysis.

```
# Subset data in the first experiment
mae[[1]] <- subsetByPrevalentFeatures(mae[[1]], rank = "Genus",
                                      prevalence = 0.2,
                                      detection = 0.001)

# rclr-transform in the first experiment
mae[[1]] <- transformAssay(mae[[1]], 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 using 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
```

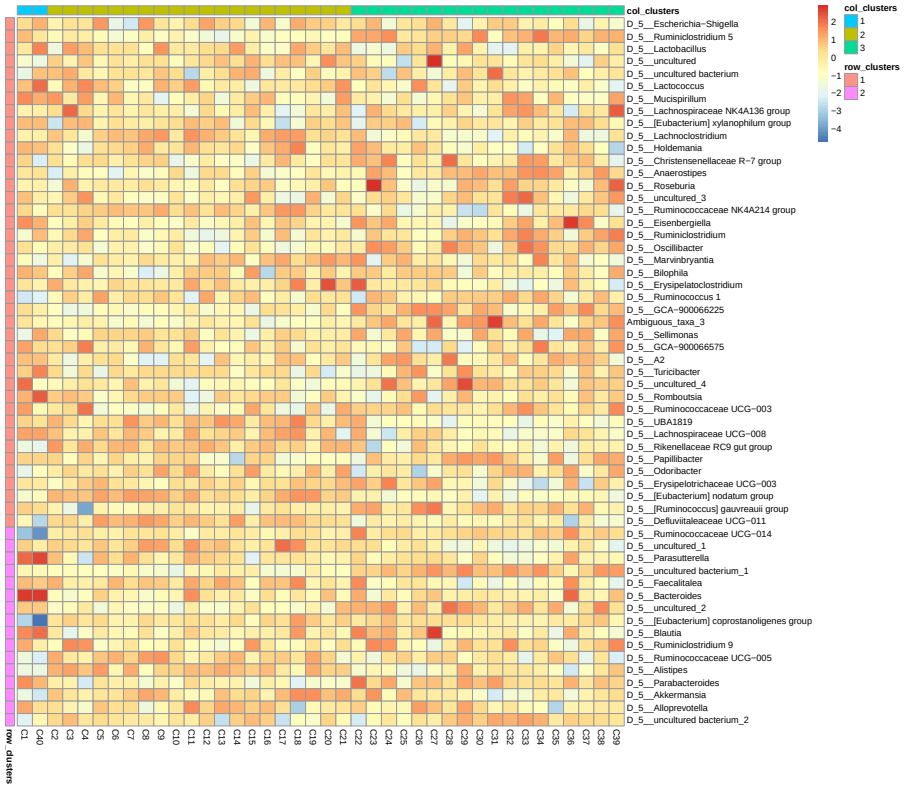
```
## [39] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1
##
## $colclass
##  C1  C2  C3  C4  C5  C6  C7  C8  C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20
##   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31 C32 C33 C34 C35 C36 C37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

```
library(pheatmap)
# z-transform for heatmap
mae[[1]] <- transformAssay(mae[[1]], assay.type = "rclr",
                           MARGIN = "features", method = "z",
                            ↪  name = "rclr_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"
```

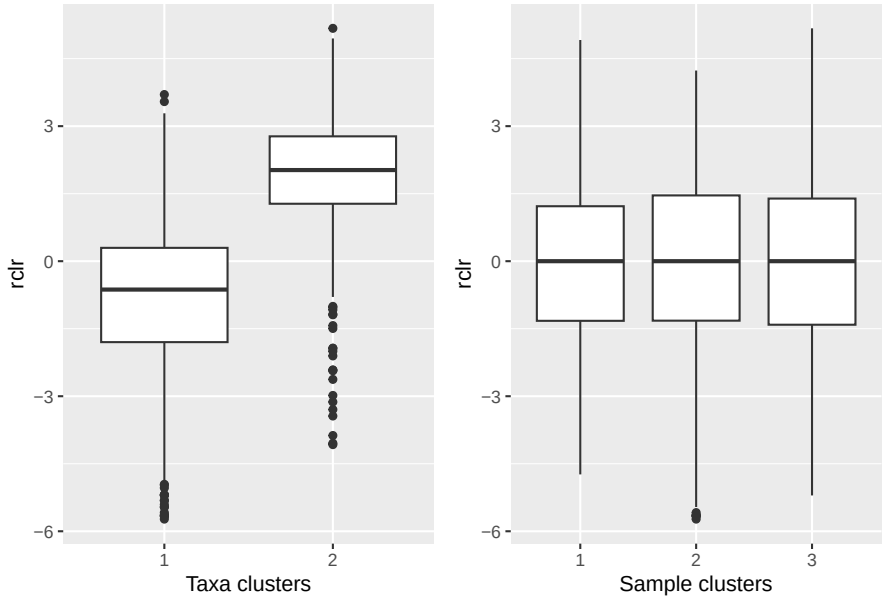
Plot the heatmap.

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



Boxplot is commonly used to summarize the results:

```
library(ggplot2)
library(patchwork)

# ggplot requires data in melted format
melt_assay <- meltAssay(mae[[1]], assay.type = "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
```



10.6.2 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. First we pre-process the data.

```
# 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 required by other steps
rownames(mae[[1]]) <- make.unique(rownames(mae[[1]]))

# Transform the metabolites to be in log basis
mae[[2]] <- transformAssay(mae[[2]], assay.type = "nmr", method =
 ↪  "log10")

# Add missing data to the metabolites
replace_na <- function(row) {
    na_indices <- which(is.na(row))
    non_na_values <- row[!is.na(row)]
    row[na_indices] <- sample(non_na_values, length(na_indices),
 ↪  replace = TRUE)
```

```
    row
}
assay(mae[[2]], "log10") <- t(apply(assay(mae[[2]], "log10"), 1,
 ↪  replace_na))
```

Next, we compute the spearman correlation matrix.

```
# Calculate correlations
corr <- getExperimentCrossCorrelation(mae, 1, 2, assay.type1 =
 ↪  "rclr",
                                       assay.type2 = "log10", mode
                                        ↪  = "matrix",
                                       correlation = "spearman")
```

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

```
library(biclust)
# Set seed for reproducibility
set.seed(3973)

# Find biclusters
bc <- biclust(corr, method = BCPlaid(), verbose = FALSE)

bc
```

```
##
## An object of class Biclust
##
## call:
##  biclust(x = corr, method = BCPlaid(), verbose = FALSE)
##
## Number of Clusters found:  6
##
## First  5  Cluster sizes:
##                     BC 1 BC 2 BC 3 BC 4 BC 5
## Number of Rows:      11    9    6    3    2
## Number of Columns:   15   13    9    9   15
```

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.

```r
# 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 cluster_2 cluster_3 cluster_4 cluster_5
## D_5__Escherichia-Shigella     FALSE     FALSE     FALSE     FALSE     FALSE
## D_5__Ruminiclostridium 5       TRUE     FALSE      TRUE     FALSE     FALSE
## D_5__Lactobacillus            FALSE     FALSE     FALSE     FALSE     FALSE
## D_5__uncultured               FALSE     FALSE     FALSE     FALSE     FALSE
## D_5__uncultured bacterium     FALSE     FALSE     FALSE     FALSE     FALSE
## D_5__Lactococcus              FALSE     FALSE     FALSE     FALSE     FALSE
##                           cluster_6 cluster_7
## D_5__Escherichia-Shigella     FALSE      TRUE
## D_5__Ruminiclostridium 5      FALSE     FALSE
## D_5__Lactobacillus            FALSE      TRUE
## D_5__uncultured                TRUE     FALSE
## D_5__uncultured bacterium     FALSE      TRUE
## D_5__Lactococcus              FALSE      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.type1,
 ↪  assay.type2, clusters1, clusters2) {
    list <- list()
```

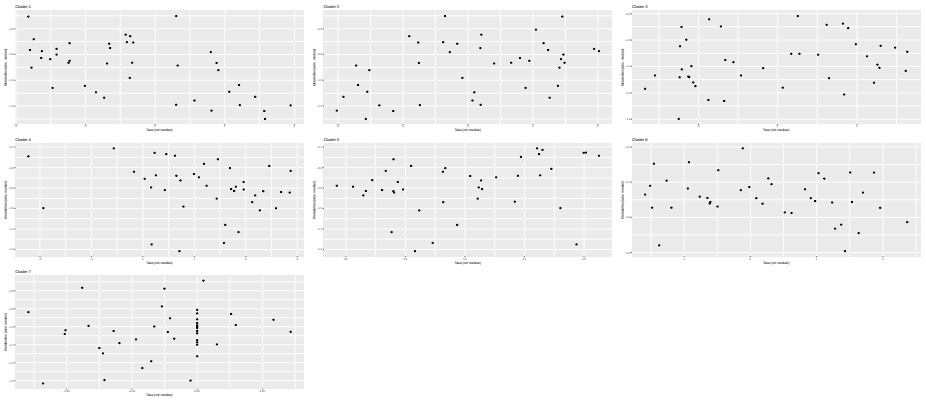
```
    # Create a data frame that includes all the information
    for (i in 1:ncol(clusters1)) {
        # Subset data based on cluster
        tse_subset1 <- tse1[clusters1[, i], ]
        tse_subset2 <- tse2[clusters2[, i], ]
        # Get assay
        assay1 <- assay(tse_subset1, assay.type1)
        assay2 <- assay(tse_subset2, assay.type2)
        # 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", "log10",
 ↪  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.

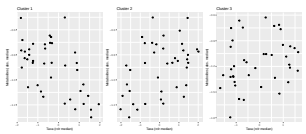
From the plots, we can see that there is low negative correlation in both cluster 1 and 3. This means that when abundance of bacteria belonging to cluster 1 or 3 is higher, the concentration of metabolites of cluster 1 or 3 is lower, and vice versa.

```
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]]
```



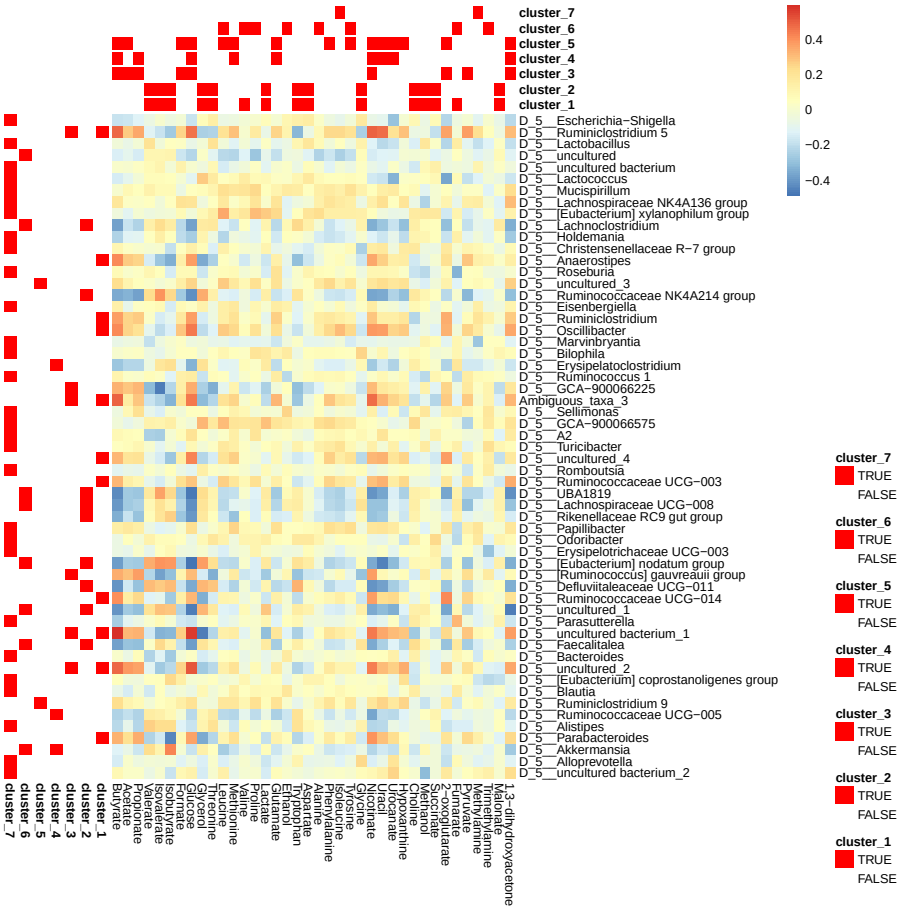
pheatmap does not allow boolean values, so they must be converted into factors.

```
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)
```



10.7 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 introduction to Differential Abundance Analysis (DAA), which is used to identify differences in the abundances of individual taxa (at any taxonomic level) between two or more groups, such as treatment and control. Here, we demonstrate its implementation on peerj13075, one of the mia datasets.

The goal of DAA is to identify biomarkers of a certain phenotype or condition, and gain understanding of a complex system by looking at its isolated components. For example, the identification of a bacterial taxon that is differentially abundant between healthy patients and diseased patients can lead to important insights into the pathophysiology of the disease. In other words, differentially abundant taxa can be involved in the dynamics of the disease, which in turn helps understand the system as a whole. Despite its relevance in current research, the DAA approach has also been subject to debate (Quinn et al., 2021).

11.1.1 Examples and tools

Due to the size and complexity of microbiome sequencing data, in this field DAA still faces several statistical challenges (Yang and Chen, 2022). These include:

- High variability. The abundance of a specific taxon could range over several orders of magnitude.
- Zero-inflation. In a typical microbiome dataset, more than 70% of the values are zeros. Zeros could be due to either physical absence (structural zeros) or insufficient sampling effort (sampling zeros).

- Compositionality. Increase or decrease in the (absolute) abundance of one taxon at the sampling site will lead to apparent changes in the relative abundances of other taxa in the sample.

As a consequence, the following approaches have been developed:

- Over-dispersed count models has been proposed to address zero inflation, such as the negative binomial model used by edgeR (Chen et al., 2016) and DESeq2 (Love et al., 2014), the beta-binomial model used by corncorb (Martin et al., 2021).
- Zero-inflated mixture models has aslo been proposed to address zero inflation, such as zero-inflated log-normal/normal mixture model used by metagenomeSeq (Paulson et al., 2017) and RAIDA (Sohn et al., 2015), zero-inflated beta-binomial model used by ZIBB (Hu et al., 2018), and zero-inflated negative binomial model used by Omnibus (Chen et al., 2018).
- Bayesian methods have been used to impute the zeros for methods working on proportion data, accounting for sampling variability and sequencing depth variation. Examples include ALDEx2 (Gloor et al., 2016) and eBay (Liu et al., 2020).
- Other methods use the pseudo-count approach to impute the zeros, such as MaAsLin2 (Mallick et al., 2020) and ANCOMBC (Lin and Peddada, 2020).
- Different strategies have been used to address compositional effects, including:
  - Robust normalization. For example, trimmed mean of M-values (TMM) normalization used by edgeR, relative log expression (RLE) normalization used by DESeq2 (Love et al., 2014), cumulative sum scaling (CSS) normalization used by metagenomeSeq, centered log-ratio transformation (CLR) normalization used by ALDEx2 (Gloor et al., 2016) and geometric mean of pairwise ratios (GMPR) normalization used by Omnibus (Chen et al., 2018). Wrench normalization (Kumar et al., 2018) corrects the compositional bias by an empirical Bayes approach, which has been recommended in metagenomeSeq (Paulson et al., 2017) tutorial.
  - Reference taxa approach used by DACOMP (Brill et al., 2022) and RAIDA (Sohn et al., 2015).
  - Analyzing the pattern of pairwise log ratios, such as ANCOM (Mandal et al., 2015).
  - Bias-correction used by ANCOMBC (Lin and Peddada, 2020).

Some of the popular tools for differential abundance analysis include:

- ALDEx2 (Gloor et al., 2016)
- ANCOMBC (Lin and Peddada, 2020)
- 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., 2022b)
- ZicoSeq (Yang and Chen, 2022)
- LDM (Hu and Satten, 2020)
- RAIDA (Sohn et al., 2015)
- DACOMP (Brill et al., 2022)
- Omnibus (Chen et al., 2018)
- eBay (Liu et al., 2020)
- ZINQ (Ling et al., 2021)
- ANCOM (Mandal et al., 2015)
- fastANCOM (Zhou et al., 2022a)
- t-test
- Wilcoxon test

We recommend to have a look at Nearing et al. (2022) who compared all these methods across 38 different datasets. Because different methods use different approaches (parametric vs non-parametric, different normalization techiniques, assumptions etc.), the results may differ between methods, sometimes substantially as Nearing et al. (2022) pointed out. More recently Yang and Chen (2022) comprehensively evaluated these methods via a semi-parametric framework and 106 real datasets. Yang and Chen (2022) also concluded that different DA methods can sometimes produce discordant results, opening to the possibility for cherry-picking tools in favor of one's own hypothesis. Therefore it is highly recommended to pick several methods to assess how robust and potentially reproducible your findings are with different methods.

In this section we demonstrate the use of four methods that can be recommended based on recent literature (ANCOM-BC (Lin and Peddada, 2020), *ALDEx2* (Gloor et al., 2016), *Maaslin2* (Mallick et al., 2020), *LinDA* (Zhou et al., 2022b) and *ZicoSeq* (Yang and Chen, 2022)).

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. Or your dataset is repeated sampling design, matched-pair design or the general longitudianl design. We will demonstrate how to include covariates in those models. 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 examples by only including two of the three groups.

```
library(mia)
library(patchwork)
library(tidySummarizedExperiment)
library(knitr)
library(tidyverse)
library(ALDEx2)
library(Maaslin2)
library(MicrobiomeStat)
library(ANCOMBC)
library(GUniFrac)

# 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)
tse0 <- peerj13075
tse0 <- tse0[ ,tse0$Geographical_location %in% c("Pune",
 ↪  "Nashik")]
# Let us make this a factor
tse0$Geographical_location <- factor(tse0$Geographical_location)

# how many observations do we have per group?
as.data.frame(colData(tse0)) %>%
count(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 some data manipulation.

Let us here do aggregation to genus level, add relative abundance assay, and perform prevalence filtering.

```
tse <- agglomerateByRank(tse0, rank = "genus") %>%
       transformAssay(assay.type = "counts",
                      method = "relabundance",
               MARGIN = "samples") %>%
       # subset based on the relative abundance assay
        ↪
       subsetByPrevalentFeatures(detection = 0,
                            prevalence = 10/100,
                 assay.type = "relabundance")

# Add also clr abundances
tse <- transformAssay(tse, method="clr", pseudocount=1) # not
 ↪  bale to run
```

Regarding 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.

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 by default 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)
```

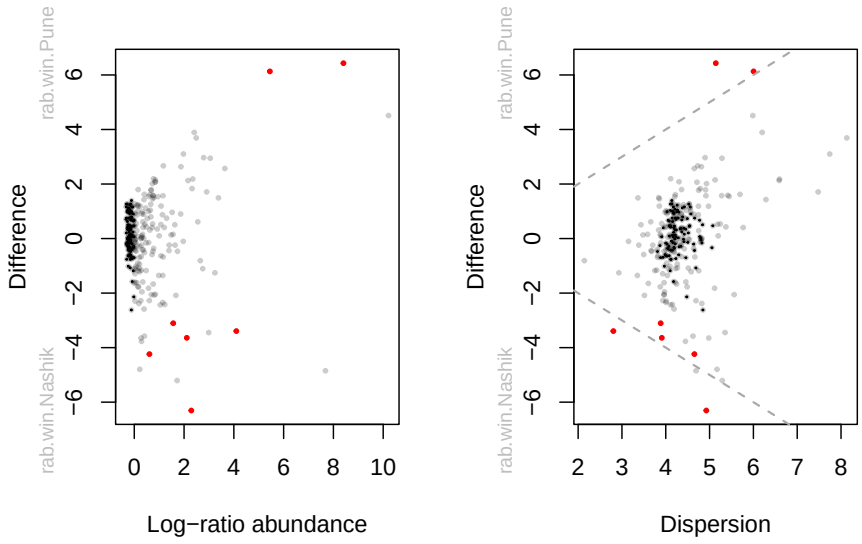
Effect sizes:

```
# 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
  dplyr::select(genus, we.eBH, wi.eBH, effect, overlap) %>%
  kable()
```

genus	we.eBH	wi.eBH	effect	overlap
Anaerococcus	0.0540	0.0150	0.9595	0.1546
Calditerricola	0.0769	0.0299	-0.7162	0.1702
Chitinivibrio	0.1216	0.0484	-0.7700	0.1776
Corynebacterium	0.0280	0.0035	1.1857	0.1037
Desulfosporomusa	0.0851	0.0359	-0.8604	0.1733
Geobacillus	0.0370	0.0081	-1.0962	0.1293
Jeotgalicoccus	0.0276	0.0251	-0.9052	0.1676
Paenibacillus	0.0837	0.0345	-0.9380	0.1932
Virgibacillus	0.1103	0.0443	-0.8750	0.1960

11.1.4 ANCOM-BC

The analysis of composition of microbiomes with bias correction (ANCOM-BC) (Lin and Peddada, 2020) 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.

```
# Agglomerate data to genus level and add this new abundance
 ↪  table to the altExp slot
altExp(tse, "genus") <- agglomerateByRank(tse, "genus")

# Identify prevalent genera
```

```
prevalent.genera <- getPrevalentFeatures(altExp(tse, "genus"),
 ↪  detection = 0, prevalence = 30/100)

# Run ANCOM-BC at the genus level and only including the
 ↪  prevalent genera

out <- ancombc2(
  data = altExp(tse, "genus")[prevalent.genera, ],
  assay_name = "counts",
  fix_formula = "Geographical_location",
  p_adj_method = "fdr",
  prv_cut = 0,
  group = "Geographical_location",
  struc_zero = TRUE,
  neg_lb = TRUE,
  global = TRUE # multi group comparison will be deactivated
   ↪  automatically
)
```

```
# store the FDR adjusted results [test on v2.0.3]
ancombc_result <- cbind.data.frame(taxid = out$res$taxon,
                        ancombc =
                         ↪  as.vector(out$res$q_Geographical_locationPune))
```

```
# store the FDR adjusted results [test on v1.2.2]
ancombc_result <- out$res %>%
 ↪  dplyr::select(starts_with(c("taxon", "lfc", "q")))
```

The object `out` contains all model output. Again, see the documentation of the function under **Value** for details. Our question whether taxa are differentially abundant can be answered by looking at the `res` object, which contains dataframes with the coefficients, standard errors, p-values and q-values. Below we show the first entries of this dataframe.

```
kable(head(ancombc_result))
```

taxon	lfc_(Intercept)	lfc_Geographical_locationPune	q_(Intercept)	q_Geographical_location
Abyssicoccus	0.0904	0.1619	0.7617	0
Acidaminococcus	0.2093	-0.2665	0.7148	0
Acinetobacter	1.5838	-1.6285	0.0861	0
Actinomyces	0.2896	-0.3084	0.6344	0
Aerococcus	-0.2045	0.6671	0.5381	0
Aeromonas	-0.3480	0.6383	0.5110	0

11.1.5 MaAsLin2

Let us next illustrate MaAsLin2 (Mallick et al., 2020). This method is based on generalized linear models and flexible for different study designs and covariate structures. For details, check their Biobakery tutorial.

```
# maaslin expects features as columns and samples as rows
# for both the abundance table as well as metadata

# We can specify different GLMs/normalizations/transforms.
# Let us use similar settings as in Nearing et al. (2021):
maaslin2_out <- Maaslin2(
  t(assay(tse)),
  data.frame(colData(tse)),
  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(maaslin2_out$results, qval <= 0.05)))
```

feature	metadata	value	coef	stderr	pval	name
Fructobacillus	Geographical_location	Nashik	0.0080	0.0011	0	Geographical_loca
Desulfosporomusa	Geographical_location	Nashik	0.0373	0.0059	0	Geographical_loca
Geobacillus	Geographical_location	Nashik	0.1294	0.0207	0	Geographical_loca
Pullulanibacillus	Geographical_location	Nashik	0.0395	0.0062	0	Geographical_loca
Chitinivibrio	Geographical_location	Nashik	0.0274	0.0045	0	Geographical_loca
Thermoanaerobacter	Geographical_location	Nashik	0.0089	0.0015	0	Geographical_loca

This will create a folder that is called like in the output specified above. It contains also figures to visualize difference between significant taxa.

11.1.6 LinDA

Lastly, we cover linear models for differential abundance analysis of microbiome compositional data (Zhou et al. (2022b)). This is very similar to ANCOMBC with few differences: 1) LinDA correct for the compositional bias differently using the mode of all regression coefficients. 2) it is faster (100x-1000x than ANCOMBC and according to the authors); 3) it supports hierarchical models. The latest ANCOMBC versions are also supporting hierarchical models. Nevertheless, LinDA seems a promising tool that achieves a very good power/fdr trade-off together with ANCOMBC according to the review. The speed improvements might make it critical especially for datasets that have higher sample or feature set sizes.

```
meta <- as.data.frame(colData(tse)) %>%
 ↪  dplyr::select(Geographical_location)
linda.res <- linda(
  as.data.frame(assay(tse)),
  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.
```

```
linda_out <- linda.res$output$Geographical_locationPune
```

```
# to scan the table for genera where H0 could be rejected:
kable(head(filter(as.data.frame(linda_out), reject)))
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj	reject	df
Acidaminococcus	1194.7	-1.9084	0.3579	-5.332	0.0000	0.0000	TRUE	45
Aciditerrimonas	393.5	-0.6655	0.2184	-3.048	0.0039	0.0165	TRUE	45
Actinomadura	836.9	-1.7985	0.3596	-5.001	0.0000	0.0001	TRUE	45
Agromyces	938.1	-1.7530	0.3910	-4.483	0.0001	0.0004	TRUE	45
Aminivibrio	416.9	-0.7489	0.2352	-3.185	0.0026	0.0128	TRUE	45
Amycolatopsis	556.1	-0.9299	0.3320	-2.801	0.0075	0.0302	TRUE	45

11.1.7 ZicoSeq

Subsequently, we add a linear model and permutation-based method, see details at tutorial.

This approach has been assessed to exhibit high power and a low false discovery rate, which has the following components:

1. Winsorization to decrease the influence of outliers;
2. Posterior sampling based on a beta mixture prior to address sampling variability and zero inflation;
3. Reference-based multiple-stage normalization to address compositional effects;

```
set.seed(123)
meta <- as.data.frame(colData(tse))
zicoseq.obj <- GUniFrac::ZicoSeq(meta.dat = meta,
                                 feature.dat =
                                  ↪  as.matrix(assay(tse)),
                                 grp.name =
                                  ↪  'Geographical_location',
                                 adj.name = NULL,
                                 feature.dat.type = 'count',
                                 prev.filter = 0,
                                 perm.no = 999,
                                 mean.abund.filter = 0,
                                 max.abund.filter = 0,
                                 return.feature.dat = T)
```

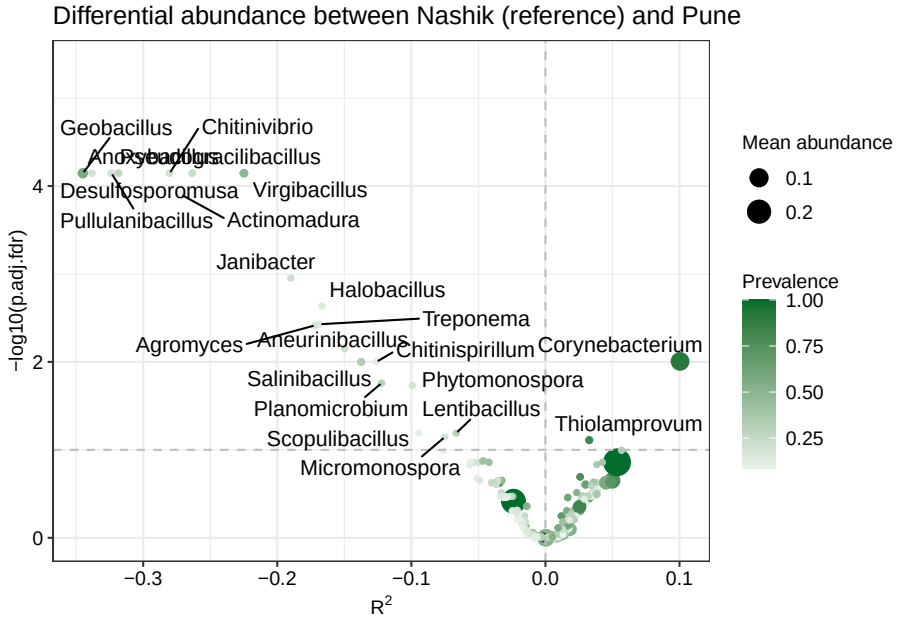
```
## 0  features are filtered!
## The data has  47  samples and  262  features will be tested!
## On average,  1  outlier counts will be replaced for each feature!
## Fitting beta mixture ...
## Finding the references ...
## Permutation testing ...
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## Completed!
```

```
zicoseq_out <- cbind.data.frame(p.raw=zicoseq.obj$p.raw,
 ↪  p.adj.fdr=zicoseq.obj$p.adj.fdr)
```

```
kable(head(filter(zicoseq_out, p.adj.fdr<0.05)))
```

	p.raw	p.adj.fdr
Actinomadura	0.001	0.0001
Agromyces	0.001	0.0038
Aneurinibacillus	0.001	0.0071
Anoxybacillus	0.001	0.0001
Chitinispirillum	0.001	0.0100
Chitinivibrio	0.001	0.0001

```
## x-axis is the effect size: R2 * direction of coefficient
ZicoSeq.plot(ZicoSeq.obj = zicoseq.obj,
             meta.dat = meta,
         pvalue.type ='p.adj.fdr')
```



11.1.8 Comparison of methods

The different methods yield somewhat different results but they could be also expected to overlap to a substantial degree. As an exercise, you can compare

the outcomes between the different methods in terms of effect sizes, significances, or other aspects that are comparable between the methods.

11.2 Confounding variables

Confounders are common in experimental research. In general, these can be classified into 3 types:

- Biological confounder, such as age, sex, etc.
- Technical confounder that caused by data collection, storage, DNA extraction, sequencing process, etc.
- Confounder caused by experimental models, such as cage effect, sample background, etc.

Adjusting confounder is necessary and important to reach a valid conclusion. 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 DESeq2, edgeR, ANCOMBC, LDM, Aldex2, Corncob, MaAsLin2, ZicoSeq, fastANCOM and ZINQ allow this. Below we show how to include a confounder/covariate in ANCOMBC, LinDA and ZicoSeq.

11.2.1 ANCOMBC

```
# perform the analysis
ancombc_cov <- ancombc2(
  data = tse,
  assay_name = "counts",
  tax_level = "genus",
  fix_formula = "Geographical_location + Age",
  p_adj_method = "fdr",
  lib_cut = 0,
  group = "Geographical_location",
  struc_zero = TRUE,
  neg_lb = TRUE,
  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.
```

```
tab <- ancombc_cov$res %>% dplyr::select(starts_with(c("taxon",
 ↪  "lfc", "q")))
kable(head(tab))
```

taxon	lfc_(Intercept)	lfc_Geographical_locationPune	lfc_AgeElderly	lfc_AgeMiddle_age	q
Abyssicoccus	0.0723	0.0319	0.3264	0.2208	
Acidaminococcus	0.2364	0.0584	-0.6752	-0.3702	
Acinetobacter	1.5023	-3.2362	2.2397	2.1993	
Actinomyces	0.3740	0.0866	-0.7335	-0.6522	
Actinoplanes	0.9135	-1.1217	-0.1497	0.0000	
Aerococcus	-0.2178	0.6432	0.0922	0.0864	

11.2.2 LinDA

```
linda_cov <- linda(
  as.data.frame(assay(tse, "counts")),
  as.data.frame(colData(tse)),
  formula = '~ Geographical_location + Age',
  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.
```

```
linda.res <- linda_cov$output$Geographical_locationPune
```

```
kable(head(filter(linda.res, reject==T)))
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj	reject	df
Acidaminococcus	1140.5	-1.438	0.4390	-3.276	0.0021	0.0182	TRUE	43
Actinomadura	804.7	-1.520	0.4477	-3.394	0.0015	0.0139	TRUE	43
Anaerococcus	939.0	6.795	1.3978	4.861	0.0000	0.0004	TRUE	43
Aneurinibacillus	988.0	-1.687	0.5426	-3.110	0.0033	0.0256	TRUE	43
Anoxybacillus	1712.8	-2.727	0.5574	-4.893	0.0000	0.0004	TRUE	43
Brachybacterium	403.0	2.060	0.7303	2.821	0.0072	0.0450	TRUE	43

11.2.3 ZicoSeq

```
set.seed(123)
zicoseq.obj <- GUniFrac::ZicoSeq(meta.dat =
 ↪  as.data.frame(colData(tse)) ,
                                  feature.dat =
                                   ↪  as.matrix(assay(tse)),
                                  grp.name =
                                   ↪  'Geographical_location',
                                  adj.name = 'Gender',
                                  feature.dat.type = 'count',
                                  prev.filter = 0,
                                  perm.no = 999,
                                  mean.abund.filter = 0,
                                  max.abund.filter = 0,
                                  return.feature.dat = T)
```

```
## 0  features are filtered!
## The data has  47  samples and  262  features will be tested!
## On average,  1  outlier counts will be replaced for each feature!
## Fitting beta mixture ...
## Finding the references ...
## Permutation testing ...
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## ..................................................................................
## Completed!
```

```
zicoseq_out <- cbind.data.frame(p.raw=zicoseq.obj$p.raw,
                                p.adj.fdr=zicoseq.obj$p.adj.fdr)
```

```
kable(head(filter(zicoseq_out, p.adj.fdr<0.05)))
```

	p.raw	p.adj.fdr
Actinomadura	0.001	0.0004
Agromyces	0.001	0.0030
Aneurinibacillus	0.001	0.0036
Anoxybacillus	0.001	0.0001
Chitinispirillum	0.001	0.0089
Chitinivibrio	0.001	0.0001

11.3 Tree-based methods

Let us next cover phylogeny-aware methods to perform group-wise associations.

11.3.1 Group-wise associations testing based on balances

For testing associations based on balances, check the philr R/Bioconductor package.

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, package="mia")
tse <- peerj13075
```

Let's first preprocess the data.

```
# Agglomerate data
tse <- agglomerateByRank(tse, rank = "order")

# Apply CLR transform
tse <- transformAssay(tse, assay.type = "counts", method = "clr",
                      MARGIN="samples", pseudocount=1)

# 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.

```
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     7  13
```

```
##      Veg      16  11
##
##                Accuracy : 0.383
##                  95% CI : (0.245, 0.536)
##     No Information Rate : 0.511
##     P-Value [Acc > NIR] : 0.971
##
##                   Kappa : -0.238
##
##  Mcnemar's Test P-Value : 0.710
##
##             Sensitivity : 0.304
##             Specificity : 0.458
##          Pos Pred Value : 0.350
##          Neg Pred Value : 0.407
##              Prevalence : 0.489
##          Detection Rate : 0.149
##    Detection Prevalence : 0.426
##       Balanced Accuracy : 0.381
##
##        'Positive' Class : Mixed
##
```

mikropml offers easier interface to caret package. However, we can also use it directly.

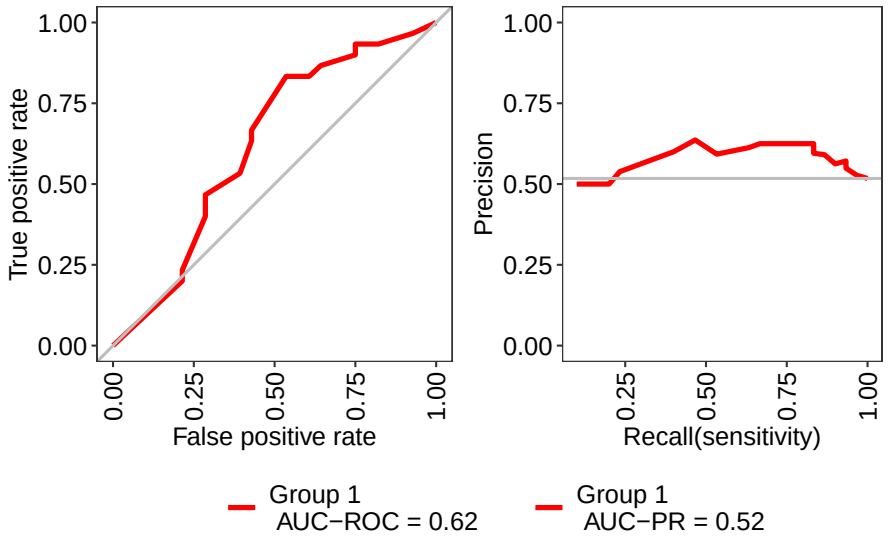
Let's use xgboost model which is another commonly used algorithm in bioinformatics.

```
# 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.

```
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

Chapter 13

Multi-assay analyses

```
library(mia)
```

Multi-omics approaches integrate data from multiple sources. For example, we can integrate taxonomic abundance profiles with metabolomic or other biomolecular profiling data to observe associations, make predictions, or aim at causal inferences. Integrating evidence across multiple sources can lead to enhanced predictions, more holistic understanding, or facilitate the discovery of novel biomarkers. In this section we demonstrate common multi-assay data integration tasks.

Cross-correlation analysis is a straightforward approach that can reveal strength and type of assocations between data sets. For instance, we can analyze if higher presence of a specific taxon relates to higher levels of a biomolecule.

The analyses can be facilitated by the multi-assay data containers, *TreeSummarizedExperiment* and *MultiAssayExperiment*. These are scalable and contain different types of data in a single container, making this framework particularly suited for multi-assay microbiome data incorporating different types of complementary data sources in a single reproducible workflow. Solutions to a number of data integration problems are discussed in more detail in Section 3. Another experiment can be stored in *altExp* slot of SE data container. Alternatively, both experiments can be stored side-by-side in a MAE data container (see sections 3.2.5 and 3.2.6 to learn more about altExp and MAE objects, respectively). Different experiments are first imported as single-assay data containers similarly to the case when only one experiment is present. After that, the different experiments can be combined into one multi-assay data container. The result is a MAE object with multiple experiments in its experiment slot, or a TreeSE object with alternative experiments in the altExp slot.

As an example, we use a dataset from the following publication: (2021) Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: associating taxonomic and metabolomic patterns in fecal microbiota with biclustering. In this study, mice were fed either with a high-fat or a low-fat diet, and with or without prebiotics, for the purpose studying whether prebiotics attenuate the negative impact of a high-fat diet on health.

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. If you like to construct the same data object from the original files instead, here you can find help for importing data into an SE object.

```
# Load the data
data(HintikkaXOData, package = "mia")
mae <- HintikkaXOData
```

```
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]__"))
```

```
# Available alternative experiments
experiments(mae)
```

```
## ExperimentList class object of length 3:
##  [1] microbiota: TreeSummarizedExperiment with 12613 rows and 40 columns
##  [2] metabolites: TreeSummarizedExperiment with 38 rows and 40 columns
##  [3] biomarkers: TreeSummarizedExperiment with 39 rows and 40 columns
```

```
# Microbiome data
getWithColData(mae, "microbiota")
```

```
## 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(6): Sample Rat ... Fat XOS
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

```
# Metabolite data
getWithColData(mae, "metabolites")
```

```
## 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(6): Sample Rat ... Fat XOS
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

```
# Biomarker data
getWithColData(mae, "biomarkers")
```

```
## 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(6): Sample Rat ... Fat XOS
## reducedDimNames(0):
## mainExpName: NULL
```

```
## altExpNames(0):
## rowLinks: NULL
## rowTree: NULL
## colLinks: NULL
## colTree: NULL
```

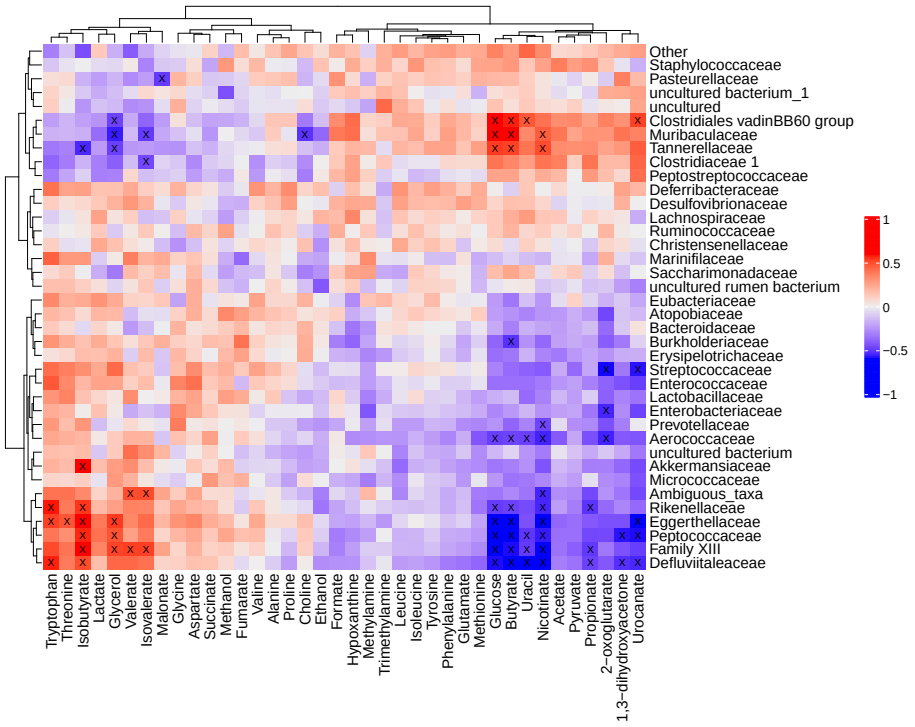
13.1 Cross-correlation Analysis

Next we can perform a cross-correlation analysis. Let us analyze if individual bacteria genera are correlated with concentrations of individual metabolites. This helps to answer the following question: "If bacterium X is present, is the concentration of metabolite Y lower or higher"?

```
# Agglomerate microbiome data at family level
mae[[1]] <- agglomerateByPrevalence(mae[[1]], rank = "Family")
# Does log10 transform for microbiome data
mae[[1]] <- transformAssay(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.type1 =
                                                ↪  "log10",
                                               assay.type2 =
                                                ↪  "nmr",
                                               method =
                                                ↪  "spearman",
                                               p_adj_threshold =
                                                ↪  NULL,
                                               cor_threshold =
                                                ↪  NULL,
                                               # Remove when mia
                                                ↪  is fixed
                                               mode = "matrix",
                                               sort = TRUE,
                                               show_warnings =
                                                ↪  FALSE)
```

Next, we create a heatmap depicting all cross-correlations between bacterial genera and metabolite concentrations.

```
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 (MOFA) is an unsupervised method for integrating multi-omic data sets in a downstream analysis (Argelaguet, 2018). 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 multiple (-omics) data sets in hand.

We use the R MOFA2 package for the analysis, and install the corresponding dependencies.

```
library(MOFA2)

# For inter-operability between Python and R, and setting Python
 ↪  dependencies,
# reticulate package is needed
library(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. We can also use Poisson or Bernoulli distributions among others.

```
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]] <- transformAssay(mae[[1]], method = "relabundance")
mae[[1]] <- transformAssay(mae[[1]], assay.type = "relabundance",
 ↪  method = "rclr")

# Transforming metabolomic data with log10
mae[[2]] <- transformAssay(mae[[2]], assay.type = "nmr",
                           MARGIN = "samples",
                           method = "log10")

# Transforming biomarker data with z-transform
mae[[3]] <- transformAssay(mae[[3]], assay.type = "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 can 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] FALSE
##
## $ard_factors
## [1] TRUE
##
## $ard_weights
## [1] TRUE
```

Training options for the model are defined in the following way:

```
train_opts <- get_default_training_options(model)
head(train_opts)
```

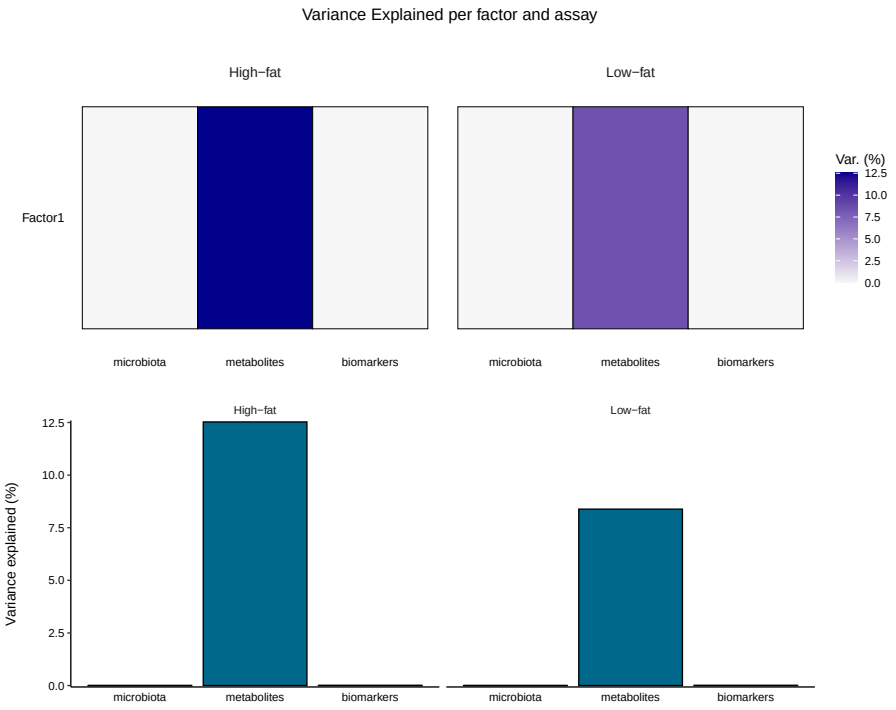
```
## $maxiter
## [1] 1000
##
## $convergence_mode
## [1] "fast"
##
## $drop_factor_threshold
## [1] -1
##
## $verbose
## [1] FALSE
##
## $startELBO
## [1] 1
##
## $freqELBO
## [1] 5
```

The model is then prepared with `prepare_mofa` and trained with `run_mofa`:

```
model.prepared <- prepare_mofa(
  object = model,
  model_options = model_opts
)

# Some systems may require the specification `use_basilisk =
 ↪  TRUE`
# so it has been added to the following code
model.trained <- run_mofa(model.prepared, use_basilisk = TRUE)
```

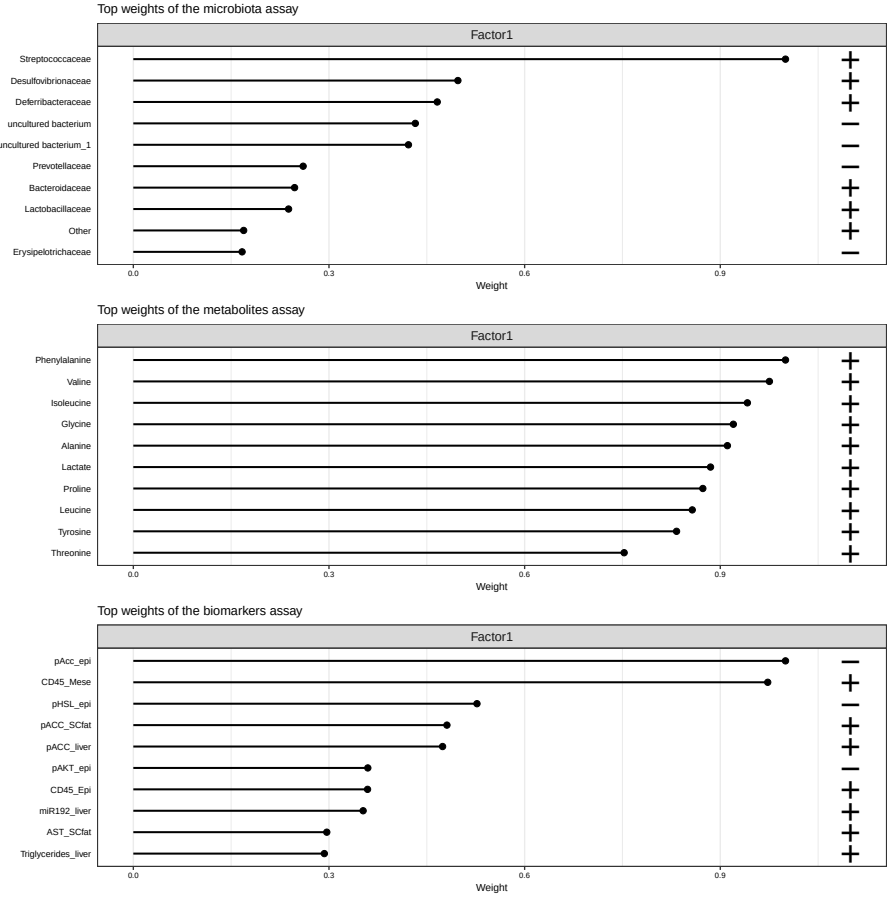
The explained variance is visualized with the `plot_variance_explained` function:

```
library(patchwork)
library(ggplot2)

plot_list <- plot_variance_explained(model.trained,
                                     x = "view", y = "factor",
                                     plot_total = T)

wrap_plots(plot_list, 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 five factors:

```
custom_plotter <- function(name) {

  p <- plot_top_weights(model.trained,
                        view = name,
                        factors = "all",
                        nfeatures = 10) +
    labs(title = paste0("Top weights of the ", name, " assay"))

}

plot_list <- lapply(c("microbiota", "metabolites", "biomarkers"),
 ↪  custom_plotter)

wrap_plots(plot_list, nrow = 3) & theme(text = element_text(size
 ↪  = 8))
```



More tutorials and examples of using the package are found at link

Chapter 14

Visualization

Data visualization will inevitably shape interpretation and motivate the next steps of the analysis. A variety of visualization methods are available for microbiome analysis but the application requires careful attention to details. Knowledge on the available tools and their limitations plays an important role in selecting the most suitable methods to address a given question.

This chapter introduces the reader to a number of visualization techniques found in this book, such as:

- barplots
- boxplots
- 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))
```

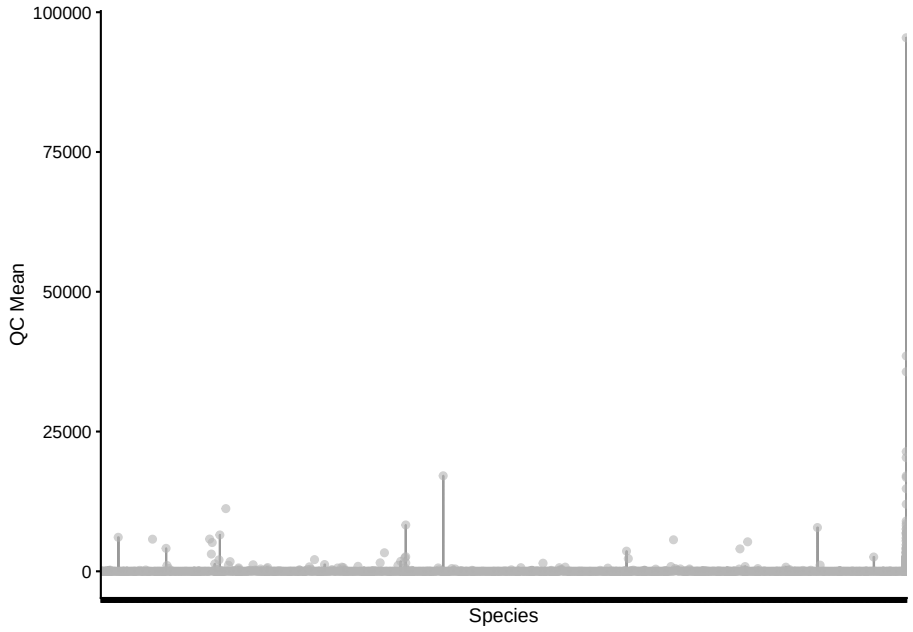
```
## [1] "Kingdom" "Phylum"  "Class"   "Order"   "Family"  "Genus"   "Species"
```

```
# list column meta data
names(colData(tse))
```

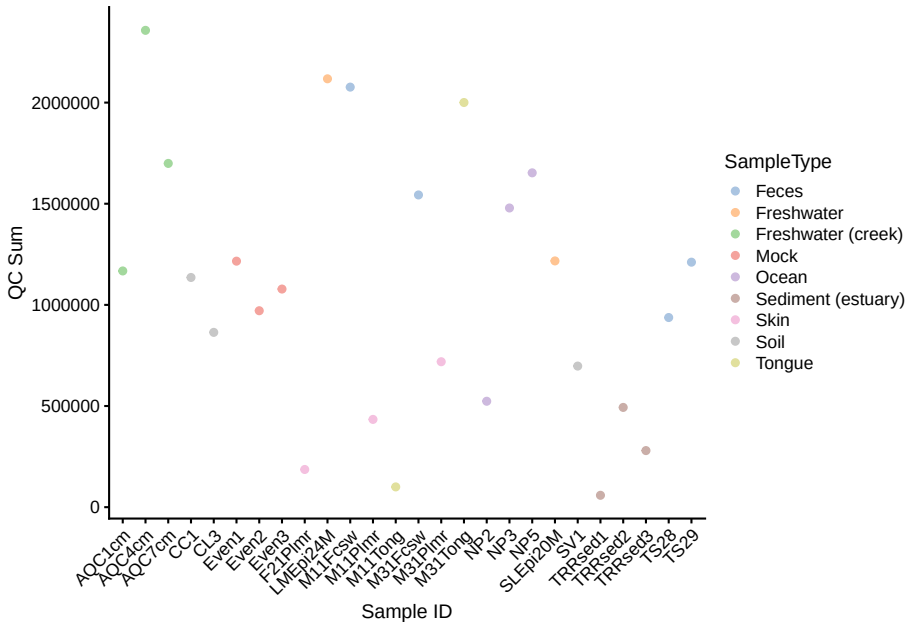
```
## [1] "X.SampleID"               "Primer"
## [3] "Final_Barcode"            "Barcode_truncated_plus_T"
## [5] "Barcode_full_length"      "SampleType"
## [7] "Description"
```

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")
```

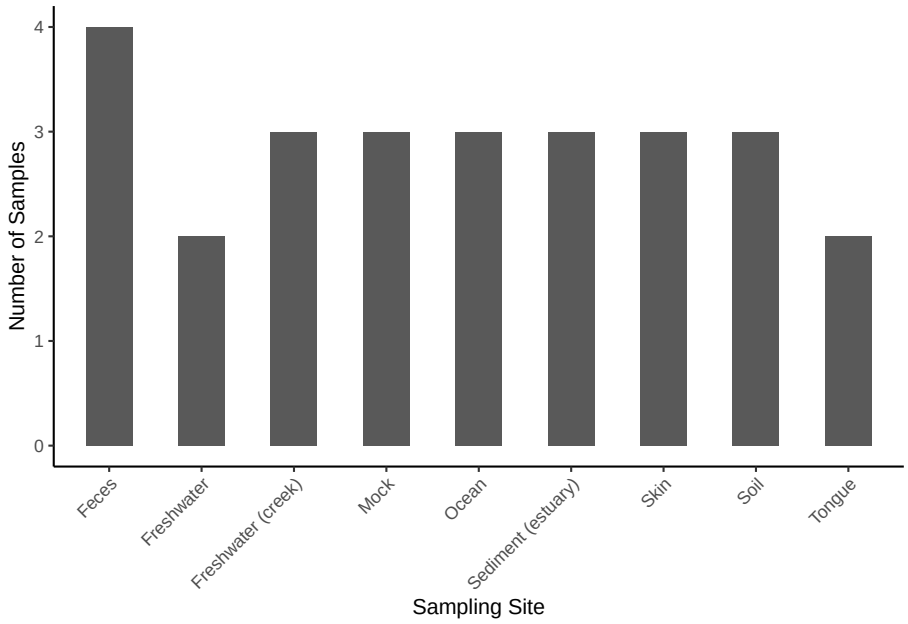



```
# plot QC Sum against Sample ID, colour-labeled by Sample Type
plotColData(tse, "sum", "X.SampleID", colour_by = "SampleType") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "Sample ID", y = "QC Sum")
```



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.

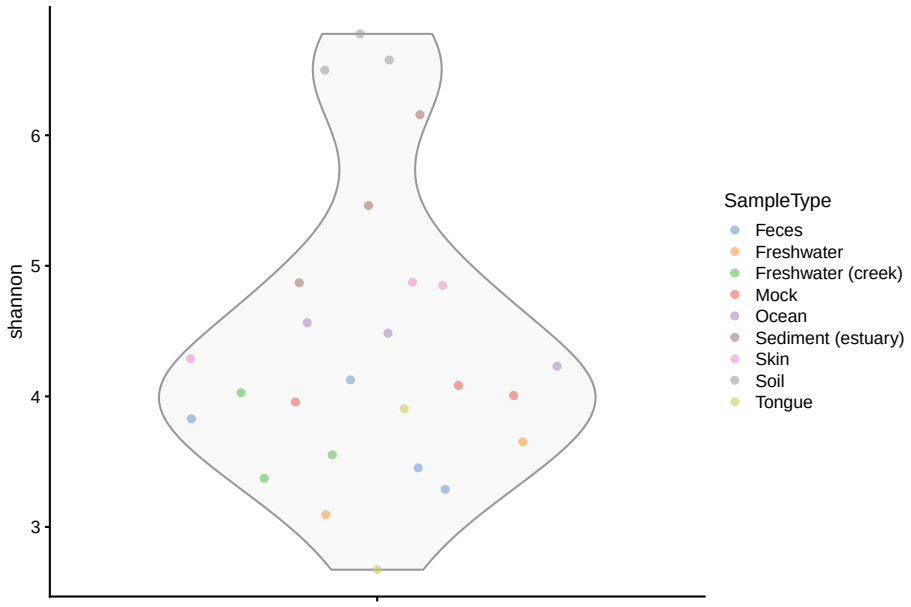
	alpha diversity	beta diversity
used metrics	diversity indices	dissimilarity indices
metric dimensionality	one-dimensional	multidimensional
suitable visualization	scatter, violin, box plots	Shepard, ordination plots

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

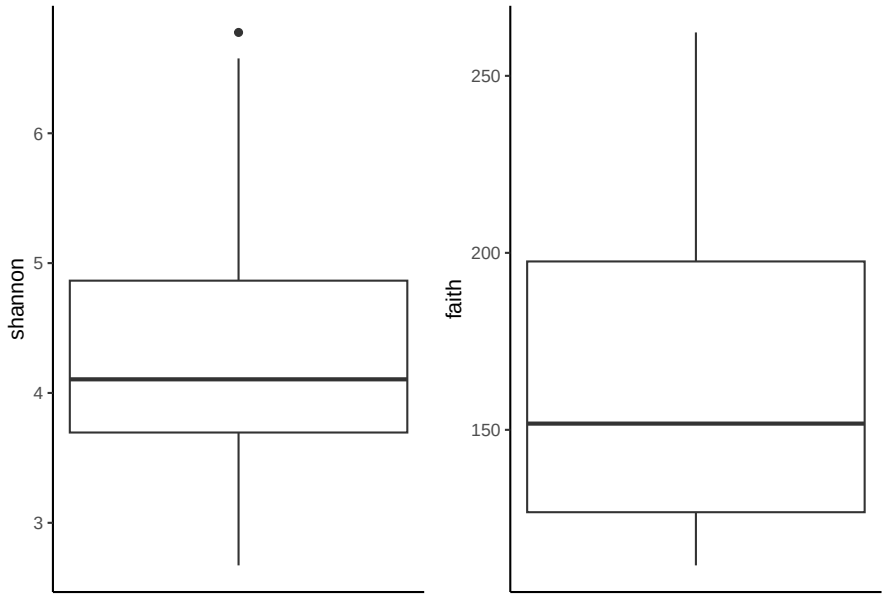
The basic method to visualize the diversity values assigned to the different samples in a `TSE` object includes the following, where each data point represents one sample:

```
# estimate shannon diversity index
tse <- mia::estimateDiversity(tse,
                              assay.type = "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.type = "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`

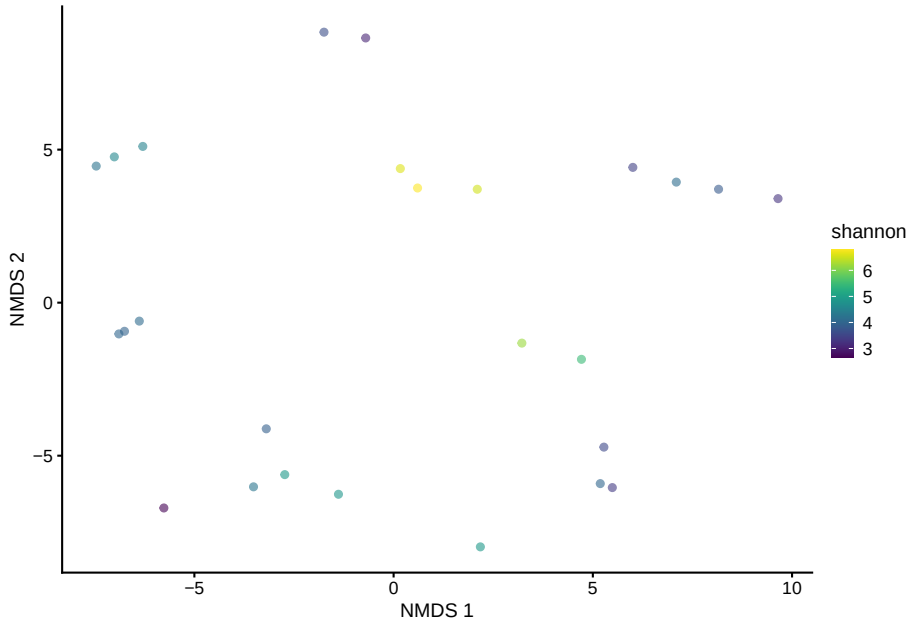
Since these clustering techniques allow for multiple coordinates or components, **coordination plots** can also span multiple dimensions, which is explained in chapter 18.

```
# 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
```

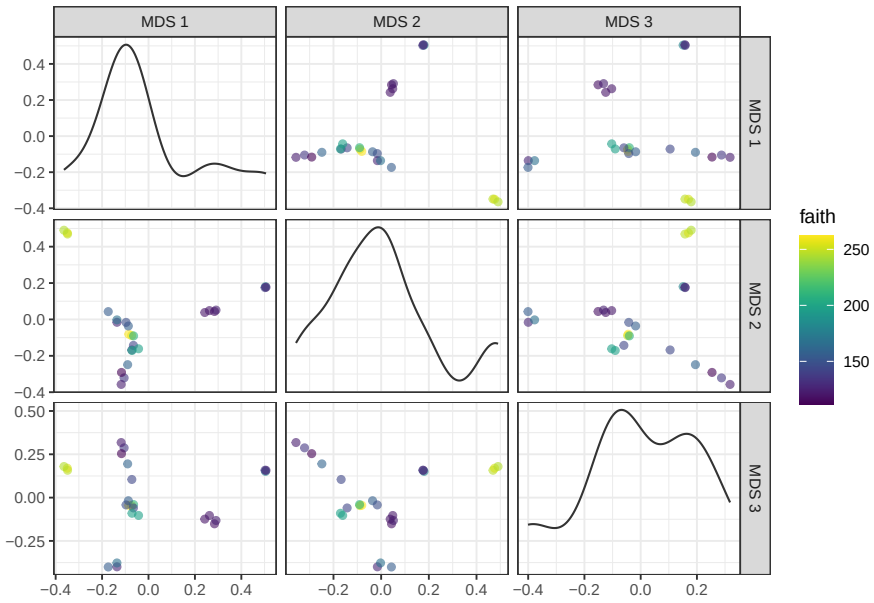
```
# plot results of a 2-component NMDS on tse,
# coloured-scaled by shannon diversity index
plotReducedDim(tse, "NMDS", colour_by = "shannon")
```



Multiple combinations of coordinates or dimensions can also be integrated into a multi-panel arrangement.

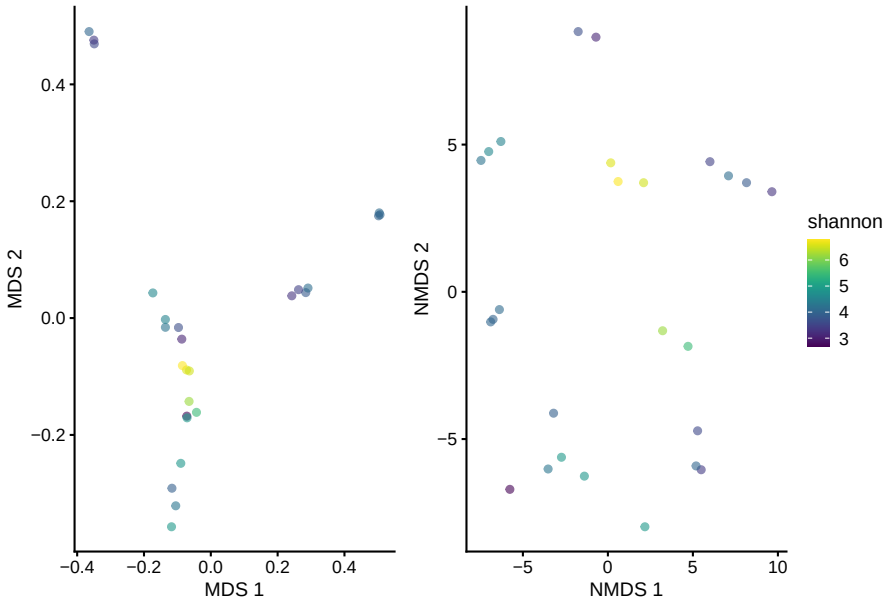
```
# perform MDS coordination method
tse <- runMDS(tse,
              FUN = vegan::vegdist,
              method = "bray",
              name = "MDS",
              assay.type = "counts",
              ncomponents = 3)
```

```
# 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

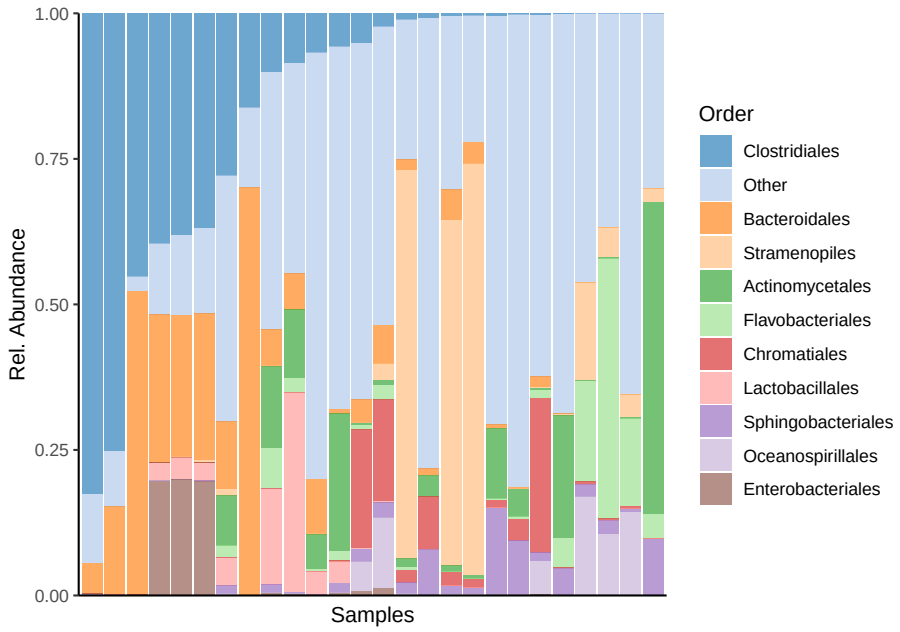
14.3.1 Heatmaps

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 <- transformAssay(tse_order,
                            assay.type = "counts",
```

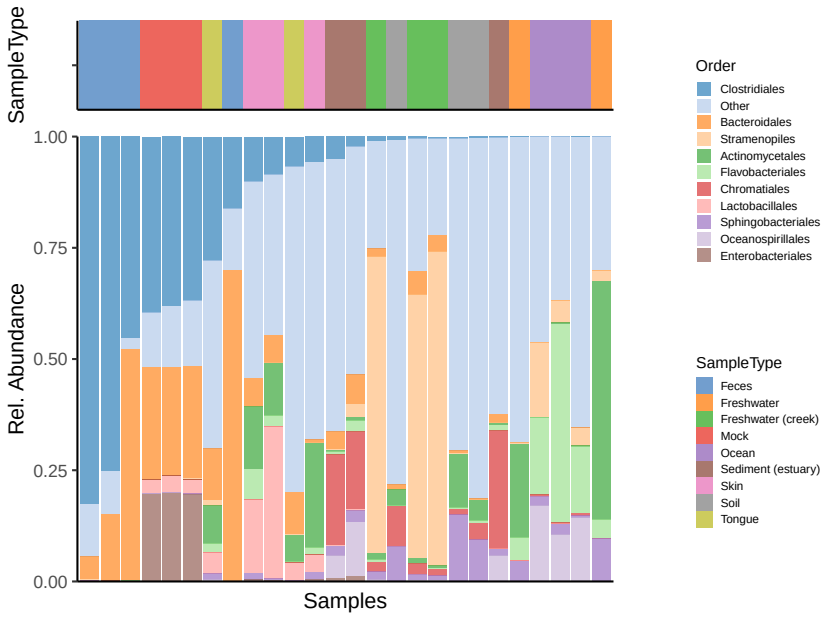
```
                                method = "relabundance")
# get top orders
top_taxa <- getTopFeatures(tse_order,
                           top = 10,
                           assay.type = "relabundance")
# leave only names for top 10 orders and label the rest with
 ↪  "Other"
order_renamed <- lapply(rowData(tse_order)$Order,
                        function(x){if (x %in% top_taxa) {x} else
                         ↪  {"Other"}})
rowData(tse_order)$Order <- as.character(order_renamed)
# plot composition as a bar plot
plotAbundance(tse_order,
              assay.type = "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.type = "relabundance",
```

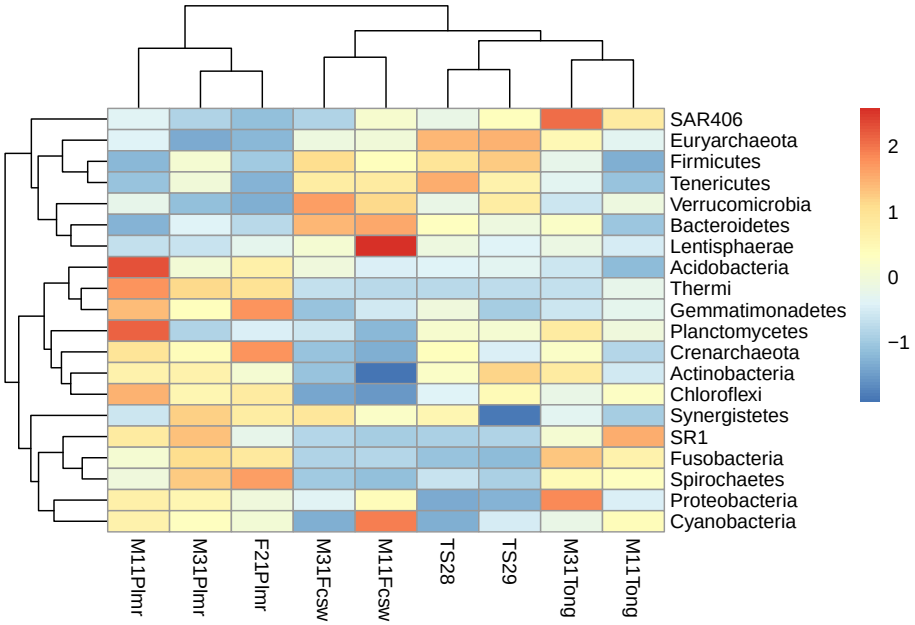
```
        rank = "Order",
            order_rank_by = "abund",
        order_sample_by = "Clostridiales",
            features = "SampleType")

# 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
library(ggpubr)
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
tse_phylum <- transformAssay(tse_phylum, MARGIN = "samples",
 ↪  method = "clr", assay.type = "counts", pseudocount=1)

# Add z-transformation on features (taxa)
tse_phylum <- transformAssay(tse_phylum, assay.type = "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
tse_phylum_subset <- transformAssay(tse_phylum_subset, method =
 ↪  "clr",
```

```
                                        MARGIN="samples",
                                        assay.type = "counts",
                                         ↪  pseudocount=1)
# Does z-transformation
tse_phylum_subset <- transformAssay(tse_phylum_subset, assay.type
 ↪  = "clr",
                                        MARGIN = "features",
                                        method = "z", name =
                                         ↪  "clr_z")

# Get n most abundant taxa, and subsets the data by them
top_taxa <- getTopFeatures(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.

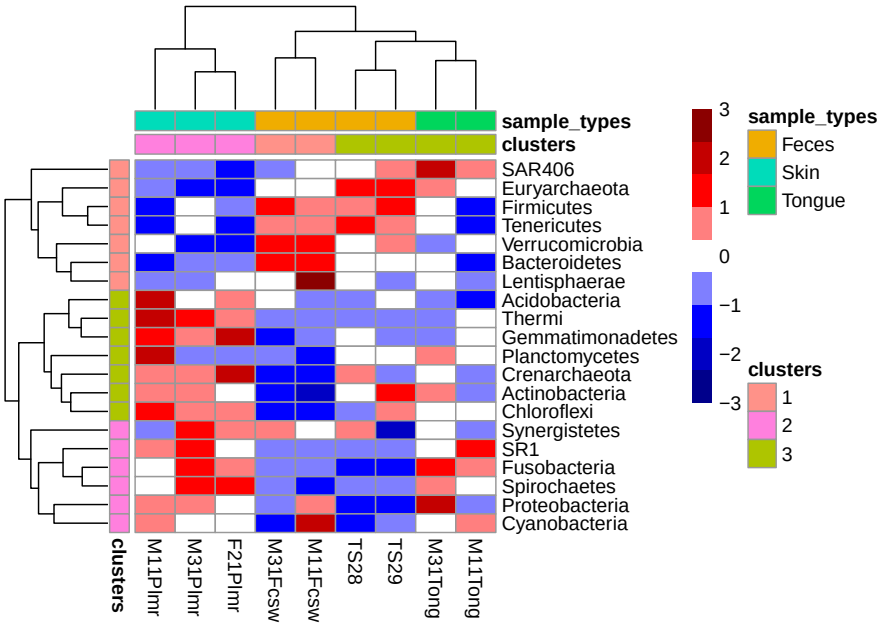
```
# Hierarchical clustering
sample_hclust <- hclust(dist(t(mat)), method = "complete")

# 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
```

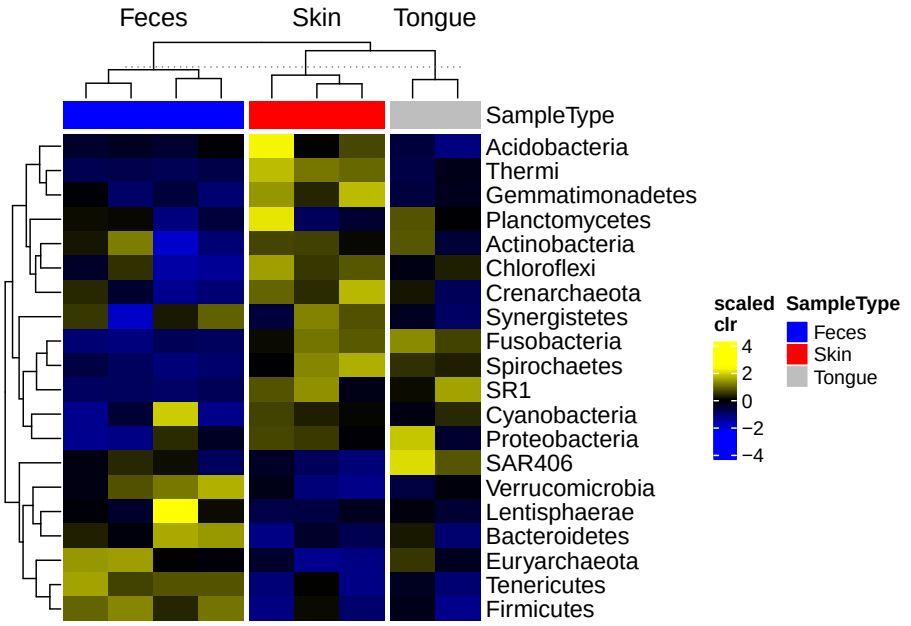
Now we can create heatmap with additional annotations.

```
# 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)
```

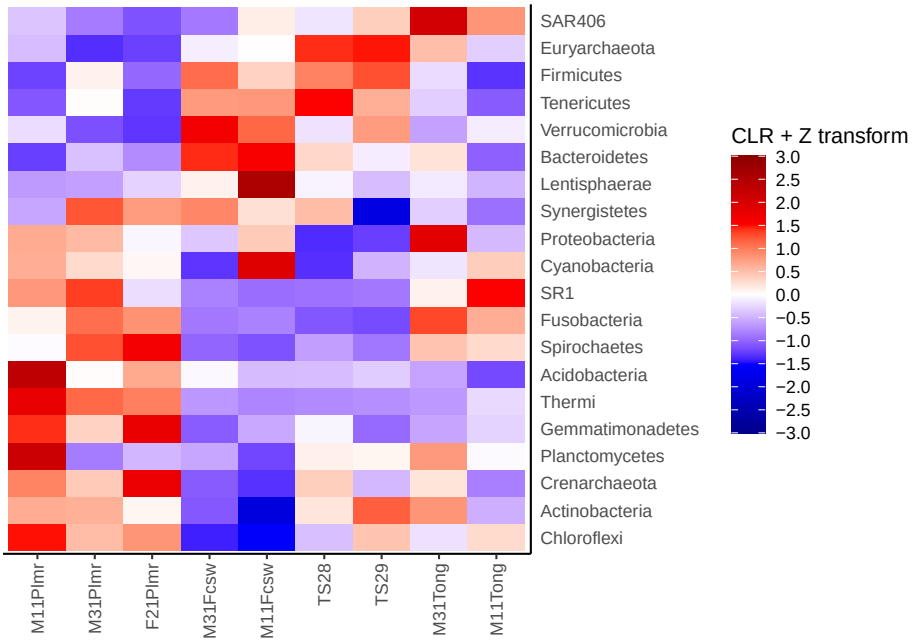


It is also possible to create an analogous heatmap by just using the *ggplot2* package. However, a relatively long code is required to generate an identical output.

```
# 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
```




```
library(patchwork)

# Create layout
design <- c(
  patchwork::area(3, 1, 4, 1),
  patchwork::area(1, 2, 1, 3),
  patchwork::area(2, 2, 2, 3),
  patchwork::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(
  patchwork::area(4, 1, 5, 1),
  patchwork::area(4, 2, 5, 2),
  patchwork::area(1, 3, 1, 4),
  patchwork::area(2, 3, 2, 4),
  patchwork::area(3, 3, 3, 4),
  patchwork::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. These are discussed further in chapters 10 and 13.

Part III

Training

Chapter 15

Training

The page provides practical information to support training and self-study.

15.1 Checklist

Brief checklist to prepare for training (see below for links).

- Install the recommended software
- Watch the short online videos and familiarize with the other available material
- Join Gitter online chat for support

15.2 Recommended software

We recommend to install and set up the relevant software packages on your own computer as this will support later use. The essential components to install include:

- R (the latest official release)
- RStudio; choose "Rstudio Desktop" to download the latest version. Check the Rstudio home page for more information. RStudio is optional.
- Install key R packages (Section 2 provides an installation script)
- After a successful installation you can consider trying out examples from Section 17 already before training. **You can run the workflows by simply copy-pasting examples.** You can then test further examples

from this tutorial, modifying and applying these techniques to your own data. Plain source code for the individual chapters of this book are available via Github

15.3 Study material

We encourage to familiarize with the material and test examples in advance:

- Short online videos on microbiome data science with R/Bioconductor
- Quarto presentations
- Other outreach material
- Orchestrating Microbiome Analysis with Bioconductor (OMA) (this book)
- Exercises for self-study
- Resources and links to complementary external material

15.4 Support and resources

For online support on installation and other matters, join us at Gitter.

You are also welcome to connect through various channels with our broader developer and user community.

15.5 Code of Conduct

We support the Bioconductor Code of Conduct. The community values an open approach to science that promotes

- sharing of ideas, code, software and expertise
- a kind and welcoming environment, diversity and inclusivity
- community contributions and collaboration

Chapter 16

Resources

16.1 Data containers

16.1.1 Data container documentation

- 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)
- MultiAssayExperiment (Ramos et al., 2017)
  - Online tutorial
  - Project page

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 (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

16.2.1 Base R and RStudio

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).

- Base R and RStudio cheatsheets
- Package-specific cheatsheets
- Visualization with ggplot2
- R graphics cookbook

16.2.2 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

16.3.1 Learn Quarto

Reproducible reporting is the starting point for robust interactive data science. Perform the following tasks:

- If you are entirely new to Quarto, take this short tutorial to get introduced to the most important functions within Quarto. Then experiment with different options from the Quarto cheatsheet.

- 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, its documentation provides guidelines to use Quarto with the R language (here) and the RStudio IDE (here).
- Further examples are tips for Quarto are available in this online tutorial to interactive reproducible reporting.

16.3.2 Additional material on Rmarkdown

Being able to use Quarto in R partly relies on your previous knowledge of Rmarkdown. The following resources can help you get familiar with Rmarkdown:

- Online tutorial
- Cheatsheet
- Documentation
- Dr. C Titus Brown's tutorial

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

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 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 knowledge on data analytics.
- 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.

17.1 Workflows

17.1.1 Reproducible reporting with Quarto

The following batch of exercises walks you through typical use cases of Quarto in RStudio. Before heading to the exercises, it is recommended to read the Quarto guidelines for RStudio

17.1.1.1 New document

This exercise gets you started with creating a Quarto document and adding text to it with typing conventions borrowed from the markdown syntax. Feel free to render the document with the **Render** button after each step to see the changes in the final report.

1. Open RStudio and create a new Quarto file named `My first Quarto`.
2. Add the subtitle `My first section` and write some text of your choice underneath. You can choose the level of headings by the number of preceding hashes (`#`).
3. Add a subsection named `List of items` and list three items underneath, both ordered and unordered. You can initialize items with numbers (`1.`, `2.`, `3.`, …) or stars (`*`) for the ordered and unordered case, respectively.
4. Add another subsection named `Link to web` and add a clickable link to the OMA book, using the pattern `[text](url)`.
5. Render the document and check its appearance

Nice start! You are now able to create a Quarto document, understand its syntax and can render it into a reproducible report. If you got stuck, you can look up the docs on creating and rendering Quarto documents.

17.1.1.2 Code chunks

While customizable text is nothing new by itself, the advantage of Quarto (and previously Rmakdown) is to combine text with code in R or other programming languages, so that both the analytical pipeline and verbal description can be put in one place. In this exercise, we learn how to write and run code in Quarto.

1. Open RStudio and create a new Quarto file.
2. Initialize a code chunk by pressing `alt` + `cmd` + `i` and define the variables `A <- "my name"` and `B <- 0` in it.
3. Write the text `Below is my first code chunk` just above the code chunk.
4. Initialize one more code chunk and add 100 to the variable `B` in it.
5. Write the text `Below I change variable B` just above the second chunk.
6. **Extra**: Write the following line of text: `my name is A and I am B years old`, where A and B are variables defined in the code chunks upstream and change if those variables are modified. Be aware that inline code can be added as `> r my_inline_command` (without `>`).

Good job. You are now able to combine text and code in a Quarto document. If you got stuck, you can refer to the Quarto docs on using code chunks.

17.1.1.3 Knitr options

Code chunks can be greatly customized in terms of visibility and execution, output size and location and much more. This is possible with the knitr chunk options, which usually appear at the beginning of the target chunk with the syntax `#| option-name: value`, also described here. In this exercise, we explore part of the knitr potential.

1. Open RStudio and create a new Quarto file.
2. Initialize three code chunks and label them as `setup`, `fig-box` and `tbl-coldata`, respectively. Remember that the name of a chunk can be specified with the `label` option.
3. Write the following code in the corresponding chunk and render the document.

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

# this line sets some options for all the chunks (global chunk
 ↪  options)
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
```

```
# fig-box
boxplot(colSums(assay(tse)) ~ tse$SampleType)
```

```
# tbl-coldata
knitr::kable(head(colData(tse)))
```

4. Set `include: false` in the `setup` chunk, `fig-width: 10` in the `fig-box` chunk and `echo: false` in the `tbl-coldata` chunk. Render the document again and find the differences from before.
5. Add the options `fig-cap` and `tab-cap` to the `fig-box` and `tbl-coldata` chunks, respectively. They require some text input, which makes for the caption of the figures or tables.
6. **Extra**: Create a cross-reference to `fig-box` and `tbl-coldata` in the text above the respective code chunk. You can do that with the syntax `@chunk-name`.
7. **Extra**: Define a custom folder for storing figures with `fig-path`. Insert it in `knitr::opts_chunk$set`, so that it applies globally to all the figures generated in the document.

Congratulations! You are now familiar with the great potential and flexibility of knitr chunk options. An exhaustive list of available options can be found in the knitr documentation.

17.1.1.4 YAML instructions

The box at the beginning of every Quarto document contains yaml options that let you define the metadata of the document. They will affect the appearance of

the document when it is rendered. By default, the box includes yaml options for the title, format and editor to be used, but much more information on layout, code execution and figures can be specified. A comprehensive list of yaml options is available here. In this exercise, we will get a tiny taste of such functionality.

1. Open RStudio and create a new Quarto file.
2. In the yaml box at the beginning of the document, change the title from `Untitled` to `My first Quarto`.
3. In the same box, add the two lines `author` and `date` followed by your name and today's date, respectively.
4. Render the document and check its appearance.
5. **Extra**: Set `toc: true` to produce a table of contents. This line should follow `format` and `html` at the second level of indentation.

Well done! Now you are able to specify yaml options and understand how they affect your Quarto document. If you got stuck, you can check this section of the Quarto documentation.

17.1.1.5 Quarto parameters

An advanced feature of Quarto consists of execution parameters, which are externally pre-defined variables that are also accessible in the Quarto document. They can be specified in the yaml box as `params`. Here we learn how to use them.

1. Open RStudio and create a new Quarto file.
2. In the yaml box at the beginning of the document, add a line named `params` followed by an indented line with `gamma: 10`
3. Initialize a code chunk and type `str(params$gamma)` in it.
4. Render the document and check what happened.
5. Define one more parameter `beta: 3` and multiply `gamma` by `beta` in a code chunk below.
6. Render the document again and check what happened.

Well done! You can now use an advanced feature of Quarto such as parameters. If you got stuck, here you can find more information about parameter definition and usage.

17.2 Data containers: TreeSE

TreeSE containers represent the working unit of the mia package. In the following exercises we learn how to construct, explore and work with them. A few demo datasets can be imported with mia and can be accessed as explained in chapter 3.3.

17.2.1 Constructing a data object

Here we cover how to construct a TreeSE from CSV files, using the components of OKeefeDSData from the microbiomeDataSets package as an example dataset.

1. Download the files with the prefix *DS* from this directory.
2. Read in the csv files with `read.csv` and store them into the variables `assays`, `rowdata` and `coldata`, respectively.
3. Create a TreeSE from the individual components with `TreeSummarizedExperiment`. Note that the function requires three arguments: assays, rowData and colData, to which you can give the appropriate item.
4. Check that importing is done correctly. E.g., choose random samples and features, and check that their values equal between raw files and TreeSE.

Usefuls functions: DataFrame, TreeSummarizedExperiment, matrix, rownames, colnames, SimpleList

17.2.2 Importing data

Raw data of different types can be imported as a TreeSE with a number of functions explained in chapter @ref(#import-from-file). You can also check the function reference in the mia package.

1. Get familiar with the microbiome data repository and read the instructions in its README to import and construct datasets from there.
2. Import data from another format (functions: loadFromMetaphlan | loadFromMothur | loadFromQIIME2 | makeTreeSummarizedExperiment-FromBiom | makeTreeSummarizedExperimentFromDADA2 …)
3. Try out conversions between TreeSE and phyloseq data containers (make-TreeSummarizedExperimentFromPhyloseq; makephyloseqFromTreeSummarizedExperiment)

17.2.3 Preliminary exploration

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. Get a summary about the TreeSE with `summary`. What is the mean count across samples? How many features recur only once (singletons)?
3. Check the dimensions of the TreeSE with `dim` or alternatively with `nrow` and `ncol`. How many samples and features are present?
4. List sample and features names with `rownames` and `colnames`.
5. Check what information about samples and features is contained by the `colData` and `rowData` of the TreeSE with `names`.
6. **Extra**: Calculate the number of unique taxa for each taxonomic rank. You can use `apply` to count unique elements for each column of rowData.

17.2.4 Assay retrieval

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. List the names of all available assays with `assayNames`.
3. Fetch the list of assays with `assays`.
4. Retrieve the first assay of the TreeSE with `assay`, where the second argument can be either the name or the index of the desired assay.

Well done! You can now locate and retrieve individual assays of a TreeSE. If you got stuck, you can refer to chapter 3.2.1 of this book.

17.2.5 Sample information

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. Check the names of the samples with `colnames`.
3. List the information on samples available in colData with `names`.
4. Visualize the colData with `View` and briefly look at the information stored in the different columns.
5. Get the abundances of all features for a specific sample, such as `ID34`. You can access sample-specific abundances with `getAbundaceSample`, specifying the desired `sample_id` and `assay.type`.

17.2.6 Feature information

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. Check the names of the features with `rownames`.
3. List the information on features available in rowData with `names`.
4. Visualize the rowData with `View` and briefly look at the information stored in the different columns.
5. Get the abundances for a specific feature, such as `OTU1810`, in all the samples. You can access feature-specific abundances with `getAbundaceFeature`, specifying the desired `feature_id` and `assay.type`.
6. **Extra**: Create a taxonomy tree based on the taxonomy mappings with `addTaxonomyTree` and display its content with `taxonomyTree` and `ggtree`.

If you got stuck, you can look up chapters @fref{datamanipulation} and 6.2.1 on how to pick specific abundances and generate row trees, respectively.

17.2.7 Other elements

Try to extract some of the other TreeSE elements listed in chapter 3. However, such data are not always included.

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. Fetch the metadata of the TreeSE. Is there any iformation available?
3. Access the phylogenetic tree with `rowTree`. How big is it in terms of tips and nodes. If you like you can visualize it with `ggtree`.
4. Check if a sample tree is available with `colTree`, which is suitable for hierarchical or nested study designs.
5. If present, obtain the information on feature DNA sequences from the DNA sequence slot.

17.3 Data manipulation

17.3.1 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

17.3.2 Library sizes

1. Calculate library sizes
2. Subsample / rarify the counts (see: subsampleCounts)

Useful functions: nrow, ncol, dim, summary, table, quantile, unique, addPerCellQC, agglomerateByRank

17.3.3 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, getPrevalentFeatures, subsetByPrevalentFeatures

17.3.4 Data exploration

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, transformAssay, ggplot, wilcox.test, agglomerateByRank, plotAbundance

17.3.5 Other functions

1. Merge data objects (merge, mergeSEs)
2. Melt the data for visualization purposes (meltAssay)

17.3.6 Assay transformation

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.
2. Transform the counts assay into relative abundances with `transformAssay` and store it into the TreeSE as an assay named `relabund` (see chapter **??**assay-transform)).
3. Similarly, perform a clr transformation on the counts assay with a `pseudocount` of 1 and add it to the TreeSE as a new assay.
4. List the available assays by name with `assays`.
5. Access the clr assay and select a subset of its first 100 features and 10 samples. Remember that assays are subsettable with `assay[row_idx, col_idx]`.
6. Take the same subset from the TreeSE, and check how this affects the individual transformed assays. TreeSE can also be subsetted with `tse[row_idx, col_idx]`.
7. **Extra**: If the data has phylogenetic tree, perform the phILR transformation.

17.4 Abundance tables

17.4.1 Taxonomic levels

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.

2. List the available taxonomic ranks in the data with `taxonomyRanks`.
3. Agglomerate the data to Phylum level with `agglomerateByRank` and the appropriate value for `Rank`.
4. Report the dimensions of the TreeSE before and after agglomerating. You can use `dim` for that.
5. **Extra**: Perform CLR transformation on the data. Does this affect agglomeration?
6. **Extra**: List full taxonomic information for a few selected taxa, such as `OTU1` and `OTU1368`. For that you can use `mapTaxonomy` on a specific subset of the TreeSE.

17.4.2 Alternative experiments

1. Import the mia package, load GlobalPatterns with `data` and store it into a variable named `tse`.
2. Check the taxonomic ranks of the features with `taxonomyRanks`. What is the deepest taxonomic rank available?
3. Agglomerate the TreeSE to each taxonomic rank and store the resulting experiments as altExps. This can be performed automatically with `splitByRanks`.
4. Check the names of the generated altExps with `altExpNames` and retrieve a complete list with `altExps`.
5. Retrieve the data agglomerated by genus from the corresponding `altExp`. As for assays, you can access the desired altExp by name or index.
6. **Extra**: Split the data based on other features with `splitOn`.

17.5 Community (alpha) diversity

17.5.1 Estimation

1. Import the mia package, load GlobalPatterns with `data` and store it into a variable named `tse`.
2. Calculate multiple alpha diversity indices with `estimateDiversity` without any additional arguments.
3. Check the names of colData with `names`. Can you identify which columns contain the alpha diversity indices?
4. **Extra**: Agglomerate the TreeSE by phylum and compare the mean Shannon diversity of the original experiment with its agglomerated version. You can use `agglomerateByRank` to perform agglomeration and `mean` to calculate the mean values of the respective columns in colData.

17.5.2 Visualization

1. Import the mia and scater packages, load GlobalPatterns with `data` and store it into a variable named `tse`.
2. Calculate Shannon diversity index and Faith's phylogenetic diversity with `estimateDiversity` and the appropriate arguments for `index`.
3. Make a boxplot of Shannon diversity on the y axis and sample type on the x axis with `plotColData`.
4. Repeat the previous point with Faith's phylogenetic diversity and compare the sample distributions of the two alpha diversity indices. How greatly do they differ?
5. **Extra**: Make a scatterplot of Shannon diversity on the y axis and Faith's phylogenetic diversity on the x axis with `plotColData`. Colour the points by sample type with the appropriate optional argument.

17.5.3 Correlation

1. Import the mia and scater packages, load peerj13075 with `data` and store it into a variable named `tse`.
2. Calculate coverage and Shannon diversity index with `estimateDiversity` and the appropriate arguments for `index`.
3. Test the correlation between the two indices with `cor.test`. Remember that colData parameters are accessible with `tse$param_name`. Use Kendall tau coefficients as `method` to measure correlation. Is the correlation weak or strong, significant or not?
4. Make a scatterplot of Shannon diversity index on the y axis and coverage on the x axis. You can do that with `plotColData`. How do the two indices relate to one another?
5. **Extra**: Compute the library size of the samples by applying `colSums` to the counts assay of the TreeSE, and test the correlation of library size with Shannon diversity or coverage. Which index is more correlated with library size?

In this example, we inspected the correlation between two related variables, also known as multicollinearity, and checked the correlation to library size, which is part of quality control. However, the correlation between alpha diversity and other numerical data about samples, such as participant's age and weight, also represent an important analysis in several studies.

17.5.4 Differences between groups

1. Import the mia package, load peerj13075 with `data` and store it into a variable named `tse`.

2. Calculate the Gini-Simpson diversity with `estimateDiversity` and the appropriate argument for `index`. Set `name` to `simpson`. You will use this name to access the diversity index from colData.
3. Inspect the Diet column in the colData. Determine how the samples are grouped in terms of diet. You can see the number of unique elements in a column with `unique`.
4. Test differences in Gini-Simpson diversity between different diets with `kruskal.test`. Remember that colData parameters are accessible with `tse$param_name`.
5. Is diversity significantly different between vegan and mixed diet? To visualize that, make a boxplot of Gini-Simpson diversity on the y axis and diet on the x axis with `plotColData`.
6. **Extra**: Repeat points 3 through 5, this time for age groups. Make sure that you are using an appropriate statistical test for the number of groups and distribution.

17.6 Community similarity

17.6.1 Reduced dimensions retrieval

1. Import the mia package, load enterotype with `data` and store it into a variable named `tse`.
2. List all available reduced dimensions with `reducedDims`. At this point, no reducedDims are likely found, because we haven't created any yet.
3. Perform PCA and store its output in the TreeSE by running `tse <- runPCA(tse, assay.type = "counts")`. Note that it is required to specify the assay on which dimensionality reduction should be conducted.
4. View the names of all available reduced dimensions with `reducedDimNames`. Has something new appeared?
5. **Extra**: Access the `PCA` reducedDim object with `reducedDim` and explore its content. How are the different dimensions stored? Try to extract an array with only the values from the second dimension by indexing the object with `[ , 2]`.

17.6.2 Visualization basics with PCA

1. Import the mia and scater packages, load enterotype with `data` and store it into a variable named `tse`.
2. Perform a 3-component PCA based on the `counts` assay. You can use `runPCA` and set the optional arguments `ncomponents` and `assay.type` to the appropriate values.
3. Plot the first two dimensions of PCA with `plotReducedDim`, to which you should give the appropriate reducedDim name as the second argument.

Note that by default only the first two dimensions are shown.
4. Check which information is stored in the ColData of the TreeSE. What would be worth visualizing in our coordination plot?
5. Make the same plot again, but this time colour the observations by Enterotype. You can do that by setting `colour_by` to the appropriate colname in the colData of the TreeSE.
6. **Extra**: Plot all three dimensions of PCA with `plotReducedDim` and the optional argument `ncomponents`. Colour observations by Enterotype. Which pair of dimensions best explains the variance between Enterotypes?

17.6.3 Principal Coordinate Analysis (PCoA)

PCoA turns out to be particularly relevant for microbiome analysis, because unlike PCA it can generate reduced dimensions from distances other than Euclidean. There are several ecological distances to choose from and you can find many of them under methods in the vignettes of `vegan::vegdist`.

1. Import the mia and scater packages, load enterotype with `data` and store it into a variable named `tse`.
2. Transform the counts assay to relative abundances with `transformAssay`.
3. Perform a Multi-Dimensional Scaling (MDS) based on the relative abundance assay in terms of Bray-Curtis dissimilarity. You can use `runMDS` with the compulsory argument `FUN = vegan::vegdist`.
4. Plot the first two dimensions of PCA with `plotReducedDim`, to which you should give the appropriate reducedDim name as the second argument. Colour the observations by Enterotype with `colour_by`.
5. **Extra**: Perform MDS again with `runMDS`, but this time use Jaccard dissimilarity. The distance metric to use can be defined with the optional argument `method`, choosing from the methods in `?vegan::vegdist`. If you don't want to overwrite the reducedDim object made in point 3, set `name` to a name of your choice. Visualize and compare it to the plot from point 4.

Good job! You are now able to produce and visualize reduced dimensions of a TreeSE. `runMDS` is actually one of several algorithms for PCoA and dimensionality reduction, which you can find in section 8.1.2.

17.6.4 PERMANOVA analysis

In this exercise we focus on studying the weight of variables on the microbiome composition. Significance of each variable on beta diversity is tested with PERMANOVA (point 4) and the homogeneity assumption is also be controlled with a PERMDISP2 analysis (point 5).

1. Import the mia and vegan packages, load peerj13075 with `data` and store it into a variable named `tse`.
2. Transform the `counts` assay into relative abundances with `transformAssay`.
3. Extract the relative abundance assay, transpose it and save it into a variable named `relabund_assay`.
4. Perform PERMANOVA with `adonis2` to see how much Diet can explain the relative abundance assay (`formula = relabund_assay ~ Diet`) in terms of Bray-Curtis dissimilarity (`method = "bray"`). Also set `data = colData(tse) by = "margin"` and `permutations = 99`. What do the results tell you about Diet with respect to beta diversity?
5. **Extra**: Test homogeneity of distribution across Diet groups with `anova(betadisper(my_mat), my_groups`, where `my_mat` is the Bray-Curtis dissimilarity matrix of `relabund_assay` calculated with `vegdist(relabund_assay, "bray")` and `my_groups` is the vector of Diet values obtained from the colData of the TreeSE.

Well done! You went through testing the effect and significance of Diet on beta diversity. Keep in mind that the formula fed to `adonis2` can take more than one independent variable, so that you can also (and very often should) include covariates of your studies.

17.6.5 Redundance analysis (RDA)

Here we apply RDA, an ordination method that provides dimensions with the largest variation between the data based in the light of the specified variables (point 3). The results of RDA are usually assessed with PERMANOVA (point 5) and the homogeneity assumption should be checked as in the previous exercise. This is a relatively complex procedure, but the way this is broken down into steps below will hopefully make more sense.

1. Import the mia and vegan packages, load peerj13075 with `data` and store it into a variable named `tse`.
2. Transform the `counts` assay into relative abundances with `transformAssay`.
3. Perform RDA with `calculateRDA` to see how much Diet can explain the relative abundance assay (`formula = assay ~ Diet` and `assay.type = relabundance`) in terms of Bray-Curtis dissimilarity (`method = "bray"`).
4. Extract the RDA dimensions from the appropriate reducedDim slot with `attr(reducedDim(tse, "RDA"), "rda)` and store it into `rda`.
5. Test the effect and significance of Diet on beta diversity by PERMANOVA with `anova.cca`. Feed this function with `rda` and set `by = "margin"` and `permutations = 99`, respectively. What do the results tell you about Diet?
6. **Extra**: Check what other parameters are stored in the colData of peerj13075, add them to the formula (`formula = assay ~ Diet + ...`)

of `calculateRDA` and proceed to see how that changes the results of PERMANOVA.

Well done! You went through an RDA analysis followed by significance testing with PERMANOVA and BETADISPER2. In the next exercise we'll go deeper quantify the contributions to beta diversity.

17.6.6 Beta diversity analysis

This exercise prompts you to implement a workflow with distance-based RDA (dbRDA). You can refer to chapter 8.3.1 for a step-by-step walkthrough, which may be simplified in the future.

1. Import the mia and vegan packages, load peerj13075 with `data` and store it into a variable named `tse`.
2. Create dbRDA with Bray-Curtis dissimilarities on relative abundances. Use PERMANOVA. Can differences between samples be explained with variables of sample meta data?
3. Analyze diets' association on beta diversity. Calculate dbRDA and then PERMANOVA. Visualize coefficients. Which taxa's abundances differ the most between samples?
4. 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, transformAssay, agglomerateByRank, ggplot, plotReducedDim, vegan::adonis2

17.7 Differential abundance

17.7.1 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, transformAssay, p.adjust

17.7.2 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, transformAssay, subsetByPrevalentFeatures

17.8 Visualization

17.8.1 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, transformAssay, ggplot, plotReducedDim

17.8.2 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.

17.9 Multiomics

17.9.1 Basic exploration

Here we learn how to conduct preliminary exploration on a MAE, using HintikkaXOData as an example dataset.

1. Import the mia package, load HintikkaXOData with `data` and store it into a variable named `mae`.
2. Which experiments make up the MAE? How many samples and features are contained by each experiment? You can get a summary for all experiments with `experiments`, and check for each individual experiment with `dim`, `nrow` and `ncol`.
3. What are the names of the features and samples of the different experiments? You can see that with `rownames` and `colnames`, respectively.
4. What information is known about the samples? Remember that information about samples is stored in the `colData` of the MAE.
5. **Extra**: How do the samples of the individual experiments map to the columns of the MAE? You can find the sample mapping in the `sampleMap` of the MAE.

So far so good. You explored a MAE and its experiments, getting a taste of how information is organized in its internal structure.

17.9.2 Experiment agglomeration

Here we learn how to manipulate an experiment contained by a MAE and save the new modified version of the experiment in a suitable place (the altExp slot).

1. Import the mia package, load HintikkaXOData with `data` and store it into a variable named `mae`.
2. Agglomerate the microbiota experiment by Genus and store the output into the `altExp` slot of the microbiota experiment, with the custom name `microbiota_genus`.
3. How many features remain after agglomerating? What are their names?
4. **Extra**: create one more alternative experiment named `prevalent_microbiota_family`, which contains the microbiota experiment agglomerated by Family with a prevalence threshold of 10%. You can agglomerate and in parallel select by prevalence with `agglomerateByPrevalence`.

Good job! You agglomerated one of the experiments in the MAE and stored it as an alternative experiment.

17.9.3 Experiment transformation

We proceed with an exercise on a different type of data manipulation, that is, transformation of assays of individual experiments in the MAE.

1. Import the mia package, load HintikkaXOData with `data` and store it into a variable named `mae`.
2. What assays are contained by each individual experiment? You can check their names with `assays`.
3. Apply a log10 transformation to the assay of the metabolite experiment. For that you can use `transformAssay` and don't forget to specify the assay to be transformed with the argument `assay.type`.
4. Apply a CLR transformation to the counts assay of the microbiota experiment. To ensure non-null values in the assay, set `pseudocount` equal to 1.

You made it! You learnt how to apply different transformations to the assays of individual experiments in a MAE with `transformAssay`, specifying optional arguments based on the used method.

17.9.4 Assay extraction

The following exercise walks you through disassembling a MAE object in order to retrieve a specific assay, or to store its components as multiple separate csv files.

1. Import the mia package, load HintikkaXOData with `data` and store it into a variable named `mae`.
2. Extract the individual metabolite experiment from the MAE into a distinct TreeSE object named `metabolites`.
3. Which and how many assays are contained by `metabolites`? You can check that with `assays` or `assayNames`.
4. Write a csv file for the nmr assay with `write.csv`. You can access an individual assay of a TreeSE with `assay` by specifying the name of the desired assay.
5. **Extra**: Repeat step 1 thorugh 4 also for the microbiota and biomarkers experiments, so that a completely disassembled version of the MAE is available.
6. **Extra**: Besides experiments, MAEs also include a sampleData and a sampleMap, which are accessible with `colData(mae)` and `sampleMap(mae)`, respectively. Save also each of these two elements into a csv file.

Well done! You just splitted a MAE into its components and stored them as csv files. This script shows a possible approach.

17.9.5 MAE reconstruction

Next, we will try to reconstruct the same MAE from the files you created. Make sure you know their names and location! Alternatively, you can download this directory with the readily disassembled components of HintikkaXOData.

1. Read in the csv files containing assays with `read.csv` and save each of them into a variable named `<assay name>_assays`.
2. Create one TreeSE from each assays object with the `TreeSummarizedExperiment` function, as explained in this exercise.
3. Read in the sampleData and the sampleMap and store them into the variables `sample_data` and `sample_map`, respectively.
4. Combine the components with `MultiAssayExperiment`, where the first argument is an `ExperimentList` (for now include only the microbiota and metabolites TreeSEs), the second is colData and the third is sampleMap.
5. Make sure that the MAE experiments are identical to the original TreeSEs. You can do that qualitatively by checking their `head` and quantitatively by looking at their `dim`.
6. **Extra**: Add the biomarkers TreeSE as a new experiment to the MAE. Note that new experiments can be added to a MAE through simple concatenation with `c(mae, experiment)`.

Good job! Now you are aware of how MAEs are built and we can proceed to some analytical exercises.

17.9.6 Cross-correlation analysis

Now we will perform a cross-correlation analysis between two of the experiments in the MAE.

1. Import the mia package, load HintikkaXOData with `data` and store it into a variable named `mae`.
2. Analyze correlations between the microbiota and the biomarkers experiments with `getExperimentCrossAssociation`. Don't forget to specify the experiments you want to compare with the arguments `experiment1` and `experiment2`, and which of their assays with `assay.type1` and `assay.type2`.
3. What does the output look like? By default, correlation is measured in terms of Kendall tau coefficients. Repeat point 2, but this time change `method` to Spearman coefficients.
4. Are you able to infer significance from the output? In order to also obtain p-values from the cross-correlation analysis, repeat point 2 with the additional argument `test_significance = TRUE`.

5. Visualize results with a heatmap similarly to the example in section 13.1. Do you see any significant correlations? Interpret your results.
6. **Extra**: Perform cross-correlation analysis between the remaining experiments (microbiota vs metabolites and metabolites vs biomarkers) and visualize results with heatmaps.

Great job! You performed a cross-correlation analysis between two experiments of a MAE and visualized the results with a heatmap. You are also able to customise the correlation method and significance testing used for the analysis.

Part IV

Appendix

Chapter 18

Extra material

18.1 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
library(gtools)
library(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 <- transformAssay(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	

18.2 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.

```
library(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:

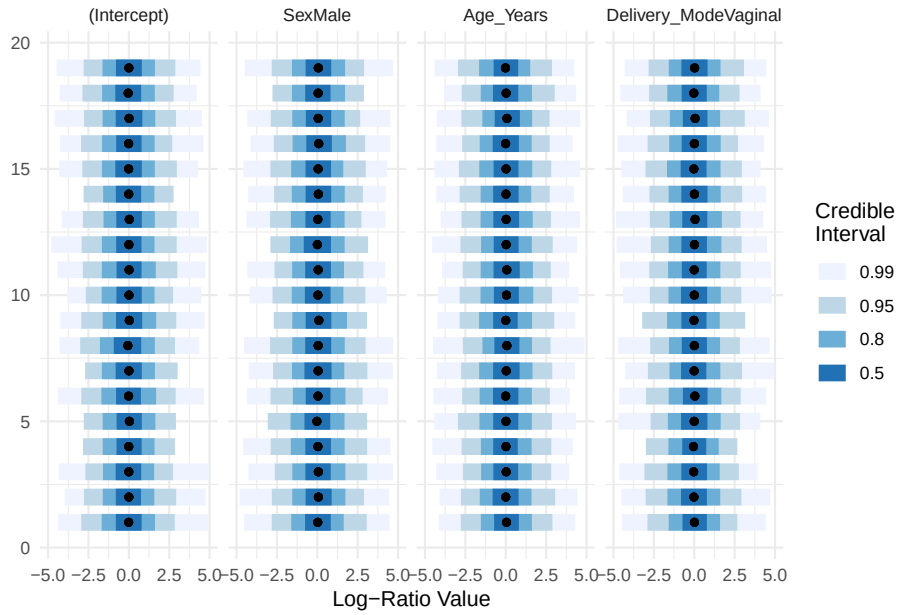
```
Y <- assays(tse_phylum)$counts
# design matrix
# taking 3 covariates
sample_data<-as.data.frame(colData(tse_phylum)[,cov_names])
X <-
 ↪  t(model.matrix(~Sex+Age_Years+Delivery_Mode,data=sample_data))
```

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
```

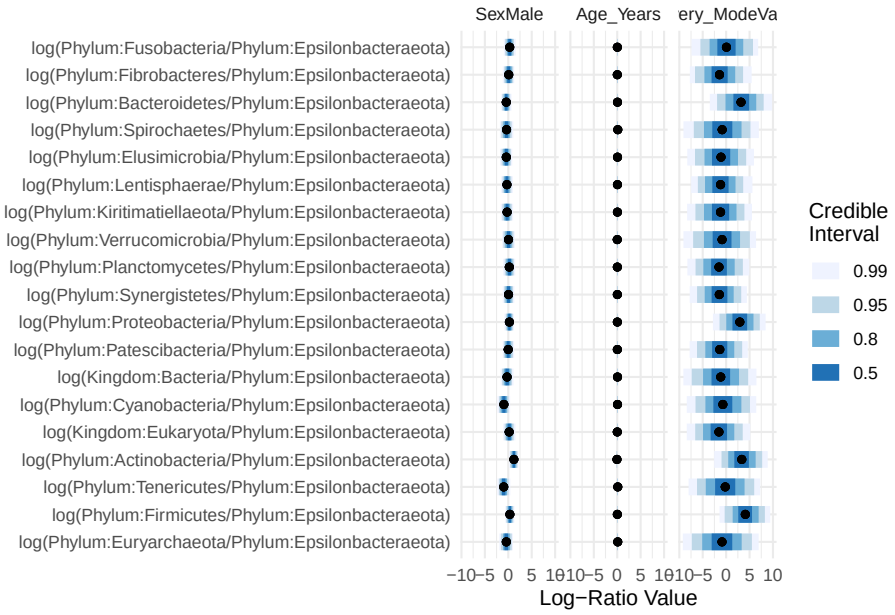
Printing a summary about the posterior:

```
ppc_summary(posterior)
```

```
## Proportions of Observations within 95% Credible Interval: 0.9979
```

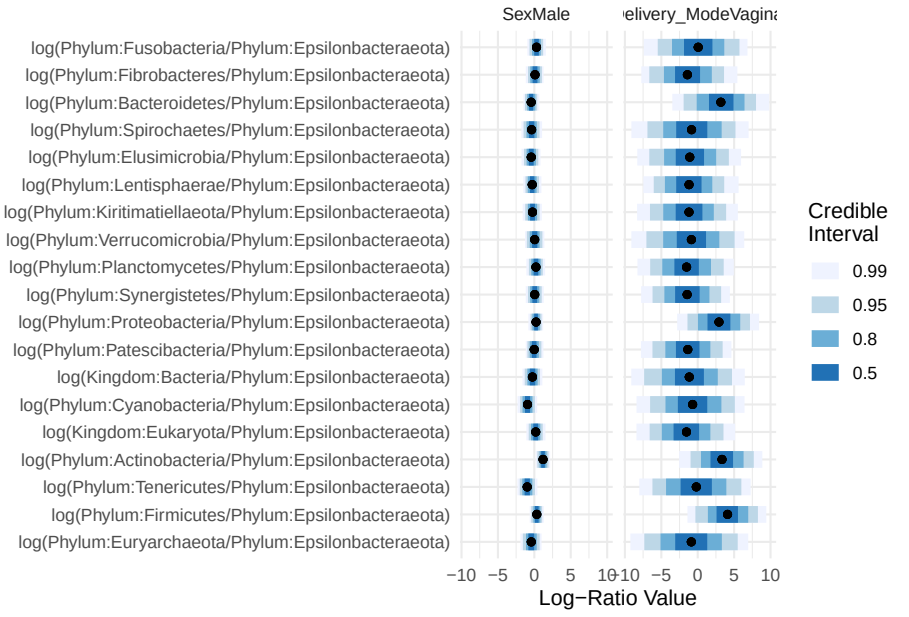
Plotting the summary of the posterior distributions of the regression parameters:

```
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)])
```



18.3 Interactive 3D Plots

```
# Installing libraryd packages
library(rgl)
library(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
library(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 <-
 ↪  addPerSampleDominantFeatures(tse_Genus,assay.type="relative_abundance",
 ↪  name = "dominant_taxa")

# Performing PCoA with Bray-Curtis dissimilarity.
tse_Genus <- runMDS(tse_Genus, FUN = vegan::vegdist, ncomponents
 ↪  = 3,
              name = "PCoA_BC", assay.type =
               ↪  "relative_abundance")

# Getting the 6 top taxa
top_taxa <- getTopFeatures(tse_Genus,top = 6, assay.type =
 ↪  "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
```

Developers

Core team

Contributions to this Gitbook from the various developers are coordinated by:

- *Leo Lahti, DSc*, professor in Data Science at the Department of Computing, University of Turku, Finland, with a focus on computational microbiome analysis. Lahti obtained doctoral degree (DSc) from Aalto University in Finland (2010), developing probabilistic machine learning with applications to high-throughput life science data integration. Since then he has focused on microbiome research and developed, for instance, the *phyloseq*-based microbiome R package before starting to develop the *TreeSummarizedExperiment / MultiAssayExperiment* framework and the mia family of Bioconductor packages for microbiome data science introduced in this gitbook. Lahti led the development of national policy on open access to research methods in Finland. He is current member in the Bioconductor Community Advisory Board and runs regular training workshops in microbiome data science.
- *Tuomas Borman*, PhD researcher and the lead developer of OMA/mia at the Department of Computing, University of Turku.

Contributors

This work is a remarkably collaborative effort. The full list of contributors is available via Github. Some key authors/contributors include:

- *Felix Ernst, PhD*, among the first developers of R/Bioc methods for microbiome research based on the *SummarizedExperiment* class and its derivatives.
- *Giulio Benedetti*, scientific programmer at the Department of Computing, University of Turku. His research interest is mostly related to Data Science. He has also helped to expand the SummarizedExperiment-based

microbiome analysis framework to the Julia language, implementing MicrobiomeAnalysis.jl.

- *Sudarshan Shetty, PhD* has supported the establishment of the framework and associated tools. He also maintains a list of microbiome R packages.
- *Henrik Eckermann*, in particular to the development of the differential abundance analyses
- *Chouaib Benchraka* provided various contributions to the package ecosystem and the OMA book
- *Yağmur Şimşek* converted the miaSim R package to support the Bioconductor framework
- *Basil Courbayre* provided various contributions to the package ecosystem and the OMA book, in particular on unsupervised machine learning
- *Matti Ruuskanen, PhD*, added machine learning techniques for microbiome analysis
- *Shigdel Rajesh, PhD*
- *Artur Sannikov*
- *Jeba Akewak*
- *Himmi Lindgren*
- *Lu Yang*

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 presented framework for microbiome data science is based on the *TreeSummarizedExperiment* data container created by Ruizhu Huang and others (Huang, 2020), (Ernst et al., 2020a), 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 brought together with other microbiome data science developers (Shetty and Lahti, 2019). Setting up the base ecosystem of packages and tutorials was then subsequently led by Tuomas Borman, Felix Ernst, and Leo Lahti. We would specifically like to thank everyone who contributed to the work supporting the *TreeSummarizedExperiment*

ecosystem for microbiome research, including but not limited to the R packages mia, miaViz, miaTime, miaSim, philr, ANCOMBC, curatedMetagenomicData, scater, scuttle, and other packages, some of which are listed in Section 2.2. A number of other contributors have advanced the ecosystem further, and will be acknowledged in the individual packages, pull requests, issues, and other work.

Ample demonstration data resources supporting this framework have been made available through the curatedMetagenomicData project by Edoardo Pasolli, Lucas Schiffer, Levi Waldron and others (Pasolli et al., 2017).

The work has drawn initial 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. 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 that have inspired this work.

In the background, 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 have advanced reproducible data science training and dissemination.

Support

This work has been supported by:

- Research Council of Finland
- FindingPheno European Union's Horizon 2020 research and innovation programme under grant agreement No 952914
- COST Action network on Statistical and Machine Learning Techniques for Human Microbiome Studies (ML4microbiome) (Moreno-Indias et al., 2021).
- Computational Life Science Research Program, Biocity Turku
- Turku University Foundation

Sessioninfo

```
sessionInfo()
```

```
## R version 4.3.0 (2023-04-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.2 LTS
##
## Matrix products: default
##
##
## 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
##
## time zone: UTC
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] BiocStyle_2.28.0 rebook_1.10.1
##
## loaded via a namespace (and not attached):
## [1] knitr_1.43      shiny_1.7.4.1   htmltools_0.5.5 rmarkdown_2.23
## [5] bookdown_0.34   miniUI_0.1.1.1  tools_4.3.0
```

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.

Brill, B., Amnon, A., and Ruth, H. (2022). Testing for differential abundance in compositional counts data, with application to microbiome studies. *The Annals of Applied Statistics*, 16.

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, J., King, E., Deek, R., Wei, Z., Yu, Y., Grill, D., and Ballman, K. (2018). An omnibus test for differential distribution analysis of microbiome sequencing data. *Bioinformatics*, 34.

Chen, Y., Lun, 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., Shetty, S., Huang, R., D.J., B., H.C., B., and Lahti, L. (2020a). The emerging r ecosystem for microbiome research. *F1000Research*, 9. A poster.

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. (2020b). *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.

Hu, T., Gallins, P., and Zhou, Y. (2018). A zero-inflated beta-binomial model for microbiome data analysis. *Stat*, 7.

Hu, Y. and Satten, G. A. (2020). Testing hypotheses about the microbiome using the linear decomposition model (ldm). *Bioinformatics*, 36:4106—4115.

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.

Kumar, M. S., Slud, V. E., Okrah, K., Hicks, C. S., Hannenhalli, S., and Bravo, C. H. (2018). Analysis and correction of compositional bias in sparse sequencing count data. *BMC genomics*, 19.

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. (2020). Analysis of compositions of microbiomes with bias correction. *Nature communications*, 11(1):1–11.

Ling, W., Ni, Z., Anna, M. P., Lenore, J. L., Anthony, A. F., Katie, A. M., and Michael, C. W. (2021). Powerful and robust non-parametric association testing for microbiome data via a zero-inflated quantile approach (zinq). *Microbiome*, 181.

Liu, T., Zhao, H., and Wang, T. (2020). An empirical bayes approach to normalization and differential abundance testing for microbiome data. *BMC Bioinformatics*, 21.

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 multiomics 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_re
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.

Sohn, M. B., Du, R., and An, L. (2015). A robust approach for identifying differentially abundant features in metagenomic samples. *Bioinformatics*, 31.

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.

Yang, L. and Chen, J. (2022). A comprehensive evaluation of microbial differential abundance analysis methods: current status and potential solutions. *Microbiome*, 10(130):2049–2618.

Zhou, C., Wang, H., Zhao, H., and Wang, T. (2022a). fastancom: a fast method for analysis of compositions of microbiomes. *Bioinformatics*, 38:2039–2041.

Zhou, H., He, K., Chen, J., and Zhang, X. (2022b). LinDA: Linear models for differential abundance analysis of microbiome compositional data. *Genome Biology*, 23(1):95.