Orchestrating Microbiome Analysis

Authors: Leo Lahti [aut], Sudarshan Shetty [aut], Felix GM Ernst [aut, cre] **Version:** 0.98.9 **Modified:** 2021-09-10 **Compiled:** 2022-04-22 **Environment:** R version 4.1.3 (2022-03-10), Bioconductor 3.14 **License:** CC BY-NC-SA 3.0 US **Copyright:** **Source:** https://github.com/microbiome/OMA

Contents

Welcome **7**

I Introduction **9**

1 Introduction **11**

2 Microbiome Data **13**

2.1 Data science framework 13

2.2 Data containers 15

2.3 Loading experimental microbiome data 22

2.4 Demonstration data 32

Session Info 34

3 Packages **37**

3.1 Package installation 37

3.2 Some available packages 37

Session Info 38

II Focus Topics **39**

4 Data Manipulation **41**

4.1 Tidying and subsetting 41

5 Exploration and quality Control **53**

5.1 Abundance 53

5.2 Prevalence 55

5.3 Quality control 60

Session Info 66

6 Taxonomic Information **69**

6.1 Assigning taxonomic information. 69

6.2 Functions to access taxonomic information 70

6.3 Data agglomeration 74

6.4 Data transformation 75

6.5 Pick specific 77

Session Info 78

7 Community diversity **81**

7.1 Estimation 82

7.2 Visualization 88

Session Info 89

8 Community similarity **91**

8.1 Explained variance 92

8.2 Community comparisons by beta diversity analysis 94

8.3 Other ordination methods 96

8.4 Visualizing the most dominant genus on PCoA 103

8.5 Further reading 108

Session Info 109

9 Community composition **111**

9.1 Visualizing taxonomic composition 111

10 Community typing **131**
10.1 Dirichlet Multinomial Mixtures (DMM) 131
10.2 Community Detection 137
10.3 Additional Community Typing 142
Session Info 142

11 Biclustering **145**
11.1 Taxa vs samples 145
11.2 Taxa vs biomolecules 149
11.3 Taxa vs taxa 155
Session Info 157

12 Differential abundance **161**
12.1 Differential abundance analysis 161
12.2 Tree-based methods 173
Session Info 177

13 Multi-assay analyses **181**
13.1 Cross-correlation Analysis 183
13.2 Multi-Omics Factor Analysis 186
Session Info 191

III Appendix **195**

14 Resources **197**
14.1 Data containers 197
14.2 R programming resources 198

15 Extra material **201**
15.1 Interactive 3D Plots 201

16 Acknowledgments **205**

Welcome

You are reading the online book, **Orchestrating Microbiome Analysis with R and Bioconductor** (Lahti et al., 2021), 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., 2020b), phyloseq tutorials (Callahan et al., 2016) and microbiome tutorials (Shetty and Lahti, 2019). This book extends these resources to teach the grammar of Bioconductor workflows in the context of microbiome data science. As such, it supports the adoption of general skills in the analysis of large, hierarchical, and multi-modal data collections. We focus on microbiome analysis tools, including entirely new, partially updated as well as previously established methods.

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

Part I

Introduction

Chapter 1

Introduction

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

We assume that the reader is already familiar with R programming. For references and tips on introductory material for R and Bioconductor, see Chapter 14. This online resource and its associated ecosystem of microbiome data science tools are a result of a community-driven development process, and welcoming new users and contributors. You can find more information on how to find us online and join the developer community through the project homepage at microbiome.github.io.

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

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

Chapter 2

Microbiome Data

2.1 Data science framework

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

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

2.2 Data containers

`SummarizedExperiment` (`SE`) 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`) was developed as an extension to incorporate hierarchical information (such as phylogenetic trees and sample hierarchies) and reference sequences.

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

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

The options and recommendations are summarized in Table 2.1.

2.2.1 Assay data

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

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

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

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

```
assays(tse)
```

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

Individual assays can be accessed via `assay`

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

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

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

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

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

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

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

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

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

2.2.2 colData

`colData` contains data on the samples.

```
colData(tse)
```

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

```
## Even3                                        Even3
```

2.2.3 rowData

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

```
rowData(tse)
```

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

2.2.4 rowTree

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

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

```
rowTree(tse)
```

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

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

```
rowLinks(tse)
```

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

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

2.2.5 Alternative experiments

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

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

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

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

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

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

```
dim(tse_phylum)
```

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

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

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

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

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

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

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

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

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

2.2.6 MultiAssayExperiments

Multiple experiments relate to complementary measurement types, such as transcriptomic or metabolomic profiling of the microbiome or the host. Multiple experiments can be represented using the same options as alternative experiments, or by using the `MultiAssayExperiment` class (Ramos and Waldron, 2020). Depending on how the datasets relate to each other the data can be stored as:

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

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

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

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

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

2.3 Loading experimental microbiome data

2.3.1 16S workflow

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

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

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

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

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

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

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

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

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

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

# Remove mock sample like it is also done in DADA2 pipeline
 ↪  tutorial
tse <- tse[ , colnames(tse) != "mock"]
```

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

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

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

2.3.2 Import from external files

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

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

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

```
samples <- read.csv(sample_file)  # Sample data (to colData)
tse <- TreeSummarizedExperiment(assays = list(counts = counts),
                                colData = samples,
                                rowData = tax)
```

Specific import functions are provided for:

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

2.3.2.1 Biom example

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

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

The data set consists of 3 files:

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

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

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

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

```
library(mia)

# Imports the data
se <- loadFromBiom(biom_file_path)

# Check
se
```

```
## class: SummarizedExperiment
## 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):
```

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

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

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

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

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

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

```
## DataFrame with 6 rows and 6 columns
##             taxonomy1          taxonomy2           taxonomy3
##           <character>        <character>         <character>
## 1726470  "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 1726471  "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 17264731 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 17264726 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
## 1726472  "k__Bacteria p__Verrucomicrobia c__Verrucomicrobiae
## 17264724 "k__Bacteria   p__Bacteroidetes      c__Bacteroidia
##                      taxonomy4              taxonomy5           taxonomy6
##                    <character>            <character>         <character>
## 1726470       o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 1726471       o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 17264731      o__Bacteroidales  f__Porphyromonadaceae g__Parabacteroides"
## 17264726      o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
## 1726472  o__Verrucomicrobiales f__Verrucomicrobiaceae     g__Akkermansia"
## 17264724      o__Bacteroidales      f__Bacteroidaceae     g__Bacteroides"
```

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

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

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

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

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

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

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

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

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

```
##                  <character>     <character>
## 1726470       Bacteroidaceae     Bacteroides
## 1726471       Bacteroidaceae     Bacteroides
## 17264731  Porphyromonadaceae Parabacteroides
## 17264726      Bacteroidaceae     Bacteroides
## 1726472  Verrucomicrobiaceae     Akkermansia
## 17264724      Bacteroidaceae     Bacteroides
```

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

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

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

Let us add a sample metadata file.

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

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

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

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

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

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

Now `colData` includes the sample metadata.

```r
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`.

```r
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).

```r
# 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:

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

2.3.3 Conversions between data formats in R

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

```r
library(mia)

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

```
GlobalPatterns_phyloseq <- GlobalPatterns
GlobalPatterns_phyloseq
```

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

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

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

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

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

```
## phyloseq-class experiment-level object
```

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

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

2.4 Demonstration data

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

2.4.1 Package data

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

List the available datasets in the `mia` package:

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

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

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

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

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

Check the documentation for this data set:

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

2.4.2 ExperimentHub data

ExperimentHub provides a variety of data resources, including the microbiomeDataSets package.

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

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

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

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

```
mae <- HintikkaXOData()
```

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

2.4.3 Other data sources

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

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

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

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

attached base packages:
```

```
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] microbiomeDataSets_1.1.5       phyloseq_1.38.0
 [3] BiocManager_1.30.16            ggplot2_3.3.5
 [5] mia_1.3.19                     MultiAssayExperiment_1.20.0
 [7] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
 [9] XVector_0.34.0                 SingleCellExperiment_1.16.0
[11] SummarizedExperiment_1.24.0    Biobase_2.54.0
[13] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[15] IRanges_2.28.0                 S4Vectors_0.32.4
[17] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[19] matrixStats_0.62.0-9000        BiocStyle_2.22.0
[21] rebook_1.4.0

loaded via a namespace (and not attached):
  [1] AnnotationHub_3.2.2           BiocFileCache_2.2.1
  [3] plyr_1.8.7                    igraph_1.3.1
  [5] lazyeval_0.2.2                splines_4.1.3
  [7] BiocParallel_1.28.3           scater_1.22.0
  [9] digest_0.6.29                 foreach_1.5.2
 [11] yulab.utils_0.0.4             htmltools_0.5.2
 [13] viridis_0.6.2                 fansi_1.0.3
 [15] magrittr_2.0.3                memoise_2.0.1
 [17] ScaledMatrix_1.2.0            cluster_2.1.3
 [19] DECIPHER_2.22.0               colorspace_2.0-3
 [21] rappdirs_0.3.3                blob_1.2.3
 [23] ggrepel_0.9.1                 xfun_0.30
 [25] dplyr_1.0.8                   crayon_1.5.1
 [27] RCurl_1.98-1.6                jsonlite_1.8.0
 [29] graph_1.72.0                  survival_3.3-1
 [31] iterators_1.0.14              ape_5.6-2
 [33] glue_1.6.2                    gtable_0.3.0
 [35] zlibbioc_1.40.0               DelayedArray_0.20.0
 [37] BiocSingular_1.10.0           Rhdf5lib_1.16.0
 [39] scales_1.2.0                  DBI_1.1.2
 [41] Rcpp_1.0.8.3                  xtable_1.8-4
 [43] viridisLite_0.4.0             decontam_1.14.0
 [45] tidytree_0.3.9                bit_4.0.4
 [47] rsvd_1.0.5                    httr_1.4.2
 [49] dir.expiry_1.2.0              ellipsis_0.3.2
 [51] pkgconfig_2.0.3               XML_3.99-0.9
 [53] scuttle_1.4.0                 CodeDepends_0.6.5
 [55] dbplyr_2.1.1                  utf8_1.2.2
 [57] AnnotationDbi_1.56.2          later_1.3.0
```

```
 [59] tidyselect_1.1.2              rlang_1.0.2
 [61] reshape2_1.4.4                munsell_0.5.0
 [63] BiocVersion_3.14.0            tools_4.1.3
 [65] cachem_1.0.6                  cli_3.2.0
 [67] DirichletMultinomial_1.36.0   generics_0.1.2
 [69] RSQLite_2.2.12                ExperimentHub_2.2.1
 [71] ade4_1.7-19                   evaluate_0.15
 [73] biomformat_1.22.0             stringr_1.4.0
 [75] fastmap_1.1.0                 yaml_2.3.5
 [77] knitr_1.38                    bit64_4.0.5
 [79] purrr_0.3.4                   KEGGREST_1.34.0
 [81] nlme_3.1-157                  sparseMatrixStats_1.6.0
 [83] mime_0.12                     compiler_4.1.3
 [85] interactiveDisplayBase_1.32.0 beeswarm_0.4.0
 [87] filelock_1.0.2                curl_4.3.2
 [89] png_0.1-7                     treeio_1.18.1
 [91] tibble_3.1.6                  stringi_1.7.6
 [93] lattice_0.20-45               Matrix_1.4-1
 [95] vegan_2.6-2                   permute_0.9-7
 [97] multtest_2.50.0               vctrs_0.4.1
 [99] pillar_1.7.0                  lifecycle_1.0.1
[101] rhdf5filters_1.6.0            BiocNeighbors_1.12.0
[103] data.table_1.14.2             bitops_1.0-7
[105] irlba_2.3.5                   httpuv_1.6.5
[107] R6_2.5.1                      promises_1.2.0.1
[109] bookdown_0.26                 gridExtra_2.3
[111] vipor_0.4.5                   codetools_0.2-18
[113] MASS_7.3-56                   assertthat_0.2.1
[115] rhdf5_2.38.1                  withr_2.5.0
[117] GenomeInfoDbData_1.2.7        mgcv_1.8-40
[119] parallel_4.1.3                grid_4.1.3
[121] beachmat_2.10.0               tidyr_1.2.0
[123] rmarkdown_2.13                DelayedMatrixStats_1.16.0
[125] shiny_1.7.1                   ggbeeswarm_0.6.0
```

Chapter 3

Packages

3.1 Package installation

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

Stable Biocondcuctor release version can be installed with:

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

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

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

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

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

3.2 Some available packages

Some of the R packages supporting the framework include:

- mia : Microbiome analysis tools

- miaViz : Microbiome analysis specific visualization
- miaSim : Microbiome data simulations
- miaTime : Microbiome time series analysis
- philr (external)

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

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

other attached packages:
[1] BiocStyle_2.22.0 rebook_1.4.0

loaded via a namespace (and not attached):
 [1] bookdown_0.26       dir.expiry_1.2.0    codetools_0.2-18
 [4] XML_3.99-0.9        digest_0.6.29       stats4_4.1.3
 [7] magrittr_2.0.3      evaluate_0.15       graph_1.72.0
[10] rlang_1.0.2         stringi_1.7.6       cli_3.2.0
[13] filelock_1.0.2      rmarkdown_2.13      tools_4.1.3
[16] stringr_1.4.0       xfun_0.30           yaml_2.3.5
[19] fastmap_1.1.0       compiler_4.1.3      BiocGenerics_0.40.0
[22] BiocManager_1.30.16 CodeDepends_0.6.5   htmltools_0.5.2
[25] knitr_1.38
```

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 the `tidyverse`, the `SE` data can be converted to long data.frame format with `meltAssay`.

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

molten_tse <- meltAssay(tse,
                        add_row_data = TRUE,
                        add_col_data = TRUE,
                        abund_values = "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>
## # ... with 499,606 more rows, and 8 more variables: Species <chr>,
## #   X.SampleID <fct>, Primer <fct>, Final_Barcode <fct>,
## #   Barcode_truncated_plus_T <fct>, Barcode_full_length <fct>,
## #   SampleType <fct>, Description <fct>
```

4.1.2 Subsetting

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

Load:

- mia
- dplyr
- knitr
- data `GlobalPatterns`

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

```r
# 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 recurrence for each value
tse$SampleType %>% table()
```

Note: after subsetting, expect the number of columns to equal the sum of the recurrences 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 recurrence for each value
rowData(tse)$Phylum %>% table()
```

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

Depending on your research question, you might need to or need not 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 the 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. For this, we use the information on the samples from the meta data `rowData(tse)`.

The first term with the `%in%` operator are includes all the features of interest, whereas the second term after the AND operator `&` filters out all the features that present a NA in place of Phylum.

```
# 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 not agglomerated data are followed.

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

# subset by feature and get rid of 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 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 not 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 get rid of 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 might occur after subsetting for instance. 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 have present 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
```

Same thing can be done for features.

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

tse_genus_sub
```

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

```
# How many bacteria there are 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 that 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 Additional functions

- mapTaxonomy
- mergeRows/mergeCols

Chapter 5

Exploration and quality Control

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

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

```
library(mia)
```

5.1 Abundance

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

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

```
# Loading data
library(microbiomeDataSets)
tse <- atlas1006()

# Add relative abundances
tse <- transformSamples(tse, method = "relabundance")

library(miaViz)
plotAbundanceDensity(tse, layout = "jitter", abund_values =
 ↪  "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", abund_values =
 ↪  "relabundance",
                     n = 5, colour_by="nationality",
                      ↪  point_alpha=1/10) +
    scale_x_log10()
```

5.2 Prevalence

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

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

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

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

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

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

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

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

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

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

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

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

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

5.2.2 Rare taxa

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

5.2.3 Plotting prevalence

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

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

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

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

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

```
altExps(tse) <- splitByRanks(tse)
altExps(tse) <-
   lapply(altExps(tse),
          function(y){
              rowData(y)$prevalence <-
                  getPrevalence(y, detection = 1/100, sort =
                   ↪  FALSE,
                                abund_values = "counts",
                                 ↪  as_relative = TRUE)
              y
          })
top_phyla <- getTopTaxa(altExp(tse,"Phylum"),
                        method="prevalence",
                        top=5L,
```

```
                          abund_values="counts")
top_phyla_mean <- getTopTaxa(altExp(tse,"Phylum"),
                             method="mean",
                             top=5L,
                             abund_values="counts")
x <- unsplitByRanks(tse, ranks = taxonomyRanks(tse)[1:6])
x <- addTaxonomyTree(x)
```

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

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

Figure 5.1: Prevalence of top phyla as judged by prevalence

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

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

5.3 Quality control

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

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

5.3.1 Top taxa

The `getTopTaxa` identifies top taxa in the data.

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

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

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

5.3.2 Library size / read count

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

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

```
## DataFrame with 26 rows and 3 columns
##                sum  detected     total
```

```
##          <numeric> <numeric> <numeric>
## CL3         864077      6964    864077
## CC1        1135457      7679   1135457
## SV1         697509      5729    697509
## M31Fcsw    1543451      2667   1543451
## M11Fcsw    2076476      2574   2076476
## ...            ...       ...       ...
## TS28        937466      2679    937466
## TS29       1211071      2629   1211071
## Even1      1216137      4213   1216137
## Even2       971073      3130    971073
## Even3      1078241      2776   1078241
```

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

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

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

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

```
library(ggplot2)

p1 <- ggplot(as.data.frame(colData(tse))) +
        geom_histogram(aes(x = sum), color = "black", fill =
         ↪  "gray", bins = 30) +
        labs(x = "Library size", y = "Frequency (n)") +
        # scale_x_log10(breaks = scales::trans_breaks("log10",
         ↪  function(x) 10^x),
        # labels = scales::trans_format("log10",
         ↪  scales::math_format(10^.x))) +
        theme_bw() +
        theme(panel.grid.major = element_blank(), # Removes the
         ↪  grid
          panel.grid.minor = element_blank(),
          panel.border = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black")) # Adds
           ↪  y-axis
```

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

library(patchwork)
p1 + p2
```

Figure 5.3: Library size distribution.

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

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

Figure 5.5: Library sizes per sample type.

5.3.3 Contaminant sequences

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

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

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

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
```

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

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

other attached packages:
 [1] patchwork_1.1.1                dplyr_1.0.8
 [3] scater_1.22.0                  scuttle_1.4.0
 [5] miaViz_1.3.3                   ggraph_2.0.5
 [7] ggplot2_3.3.5                  microbiomeDataSets_1.1.5
 [9] mia_1.3.19                     MultiAssayExperiment_1.20.0
[11] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
[13] XVector_0.34.0                 SingleCellExperiment_1.16.0
[15] SummarizedExperiment_1.24.0    Biobase_2.54.0
[17] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[19] IRanges_2.28.0                 S4Vectors_0.32.4
[21] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[23] matrixStats_0.62.0-9000        BiocStyle_2.22.0
[25] rebook_1.4.0

loaded via a namespace (and not attached):
  [1] AnnotationHub_3.2.2           BiocFileCache_2.2.1
  [3] igraph_1.3.1                  plyr_1.8.7
  [5] lazyeval_0.2.2                splines_4.1.3
  [7] BiocParallel_1.28.3           digest_0.6.29
  [9] yulab.utils_0.0.4             htmltools_0.5.2
 [11] viridis_0.6.2                 fansi_1.0.3
 [13] magrittr_2.0.3                memoise_2.0.1
 [15] ScaledMatrix_1.2.0            cluster_2.1.3
 [17] DECIPHER_2.22.0               graphlayouts_0.8.0
 [19] colorspace_2.0-3              blob_1.2.3
 [21] rappdirs_0.3.3                ggrepel_0.9.1
 [23] xfun_0.30                     crayon_1.5.1
 [25] RCurl_1.98-1.6                jsonlite_1.8.0
 [27] graph_1.72.0                  ape_5.6-2
 [29] glue_1.6.2                    polyclip_1.10-0
 [31] gtable_0.3.0                  zlibbioc_1.40.0
 [33] DelayedArray_0.20.0           BiocSingular_1.10.0
 [35] scales_1.2.0                  DBI_1.1.2
 [37] Rcpp_1.0.8.3                  viridisLite_0.4.0
 [39] xtable_1.8-4                  decontam_1.14.0
```

```
 [41] gridGraphics_0.5-1            tidytree_0.3.9
 [43] bit_4.0.4                     rsvd_1.0.5
 [45] httr_1.4.2                    dir.expiry_1.2.0
 [47] ellipsis_0.3.2                farver_2.1.0
 [49] pkgconfig_2.0.3               XML_3.99-0.9
 [51] CodeDepends_0.6.5             dbplyr_2.1.1
 [53] utf8_1.2.2                    labeling_0.4.2
 [55] ggplotify_0.1.0               tidyselect_1.1.2
 [57] rlang_1.0.2                   reshape2_1.4.4
 [59] later_1.3.0                   AnnotationDbi_1.56.2
 [61] munsell_0.5.0                 BiocVersion_3.14.0
 [63] tools_4.1.3                   cachem_1.0.6
 [65] cli_3.2.0                     DirichletMultinomial_1.36.0
 [67] generics_0.1.2                RSQLite_2.2.12
 [69] ExperimentHub_2.2.1           evaluate_0.15
 [71] stringr_1.4.0                 fastmap_1.1.0
 [73] yaml_2.3.5                    ggtree_3.2.1
 [75] knitr_1.38                    bit64_4.0.5
 [77] tidygraph_1.2.1               purrr_0.3.4
 [79] KEGGREST_1.34.0               nlme_3.1-157
 [81] sparseMatrixStats_1.6.0       mime_0.12
 [83] aplot_0.1.3                   compiler_4.1.3
 [85] beeswarm_0.4.0                filelock_1.0.2
 [87] curl_4.3.2                    png_0.1-7
 [89] interactiveDisplayBase_1.32.0 treeio_1.18.1
 [91] tweenr_1.0.2                  tibble_3.1.6
 [93] stringi_1.7.6                 highr_0.9
 [95] lattice_0.20-45               Matrix_1.4-1
 [97] vegan_2.6-2                   permute_0.9-7
 [99] vctrs_0.4.1                   pillar_1.7.0
[101] lifecycle_1.0.1               BiocManager_1.30.16
[103] BiocNeighbors_1.12.0          cowplot_1.1.1
[105] bitops_1.0-7                  irlba_2.3.5
[107] httpuv_1.6.5                  R6_2.5.1
[109] bookdown_0.26                 promises_1.2.0.1
[111] gridExtra_2.3                 vipor_0.4.5
[113] codetools_0.2-18              MASS_7.3-56
[115] assertthat_0.2.1              withr_2.5.0
[117] GenomeInfoDbData_1.2.7        mgcv_1.8-40
[119] parallel_4.1.3                ggfun_0.0.6
[121] grid_4.1.3                    beachmat_2.10.0
[123] tidyr_1.2.0                   rmarkdown_2.13
[125] DelayedMatrixStats_1.16.0     ggnewscale_0.4.7
[127] ggforce_0.3.3                 shiny_1.7.1
[129] ggbeeswarm_0.6.0
```

Chapter 6

Taxonomic Information

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

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

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

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

6.1 Assigning taxonomic information.

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

6.1.1 dada2

The dada2 package (Callahan et al., 2020) 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,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 `getUniqueTaxa` gives a list of unique taxa for the specified taxonomic rank.

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

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

6.2.1 Generate a taxonomic tree on the fly

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

```
taxonomyTree(tse)
```

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

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

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

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

6.3 Data agglomeration

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

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

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

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

```
assayNames(tse)
```

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

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

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

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

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

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

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

`altExpNames` now consists of `Family` level data. This can be extended to use any level present in Kingdom, Phylum, Class, Order, Family, Genus, Species.

6.4 Data transformation

Data transformation is a very common procedure in microbiome analysis. In transformation, each data point is replaced with transformed value that is calculated by applying transformation formula to the data point. Transformation can be used, for example, to normalize skewed data, or to reduce weight of bigger values compared to smaller values.

In mia package, transformations are applied to abundance data. The transformed abundance table is stored back to 'assays'. mia includes transformation functions for sample-wise or column-wise transformation ('transformSamples()'), and for feature-wise or row-wise transformation ('transformFeatures()').

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

```
tse <- transformSamples(x = tse, abund_values = "counts", method
 ↪  = "clr",
                        pseudocount = 1, name =
                         ↪  "clr_transformation")

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

```
##                                     CL3    CC1     SV1 M31Fcsw M11Fcsw M31Plmr
## Class:Thermoprotei               -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
## Class:Thermoprotei               -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
## Species:Sulfolobusacidocaldarius -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
## Class:Sd-NA                      -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
## Class:Sd-NA                      -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
## Class:Sd-NA                      -0.9552 -1.124 -0.7435 -0.2916 -0.2652  -0.356
##                                  M11Plmr F21Plmr M31Tong M11Tong LMEpi24M
## Class:Thermoprotei               -0.4713 -0.2645 -0.2547 -0.1572   -0.359
## Class:Thermoprotei               -0.4713 -0.2645 -0.2547 -0.1572   -0.359
## Species:Sulfolobusacidocaldarius  0.2219 -0.2645 -0.2547 -0.1572   -0.359
## Class:Sd-NA                      -0.4713 -0.2645 -0.2547 -0.1572   -0.359
## Class:Sd-NA                      -0.4713 -0.2645 -0.2547 -0.1572   -0.359
## Class:Sd-NA                      -0.4713 -0.2645 -0.2547 -0.1572   -0.359
##                                  SLEpi20M  AQC1cm  AQC4cm  AQC7cm     NP2
## Class:Thermoprotei                 0.3704  2.6250  3.7862  4.0751  0.4502
## Class:Thermoprotei                -0.3228 -0.7072  0.2697  1.1459 -0.2429
## Species:Sulfolobusacidocaldarius  -0.3228 -0.7072 -0.8289 -0.8001 -0.2429
## Class:Sd-NA                       -0.3228 -0.7072  2.3066  2.6011 -0.2429
## Class:Sd-NA                       -0.3228 -0.7072  0.2697 -0.1069 -0.2429
## Class:Sd-NA                       -0.3228 -0.7072 -0.1357  0.5862 -0.2429
##                                     NP3     NP5 TRRsed1 TRRsed2 TRRsed3    TS28
## Class:Thermoprotei               -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
## Class:Thermoprotei               -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
## Species:Sulfolobusacidocaldarius -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
## Class:Sd-NA                      -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
## Class:Sd-NA                      -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
## Class:Sd-NA                      -0.433 -0.3606 -0.2677 -0.4828 -0.4384 -0.2691
##                                     TS29   Even1   Even2   Even3
## Class:Thermoprotei               -0.2569 -0.3481 -0.2534 -0.2382
## Class:Thermoprotei               -0.2569 -0.3481 -0.2534 -0.2382
## Species:Sulfolobusacidocaldarius -0.2569 -0.3481 -0.2534 -0.2382
## Class:Sd-NA                      -0.2569 -0.3481 -0.2534 -0.2382
## Class:Sd-NA                      -0.2569 -0.3481 -0.2534 -0.2382
## Class:Sd-NA                      -0.2569 -0.3481 -0.2534 -0.2382
```

- In 'pa' transformation, 'threshold' specifies the value that divides observations to be absent or present. By default, it is 0.

```
tse <- transformFeatures(tse, method = "pa", threshold = 10)

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       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
##                                 F21Plmr M31Tong M11Tong LMEpi24M SLEpi20M
## Class:Thermoprotei                    0       0       0        0        0
## 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   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      1      1   0   0   0       0
## Class:Sd-NA                          0      0      0   0   0   0       0
## Class:Sd-NA                          0      0      0   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_transformation pa
```

6.5 Pick specific

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

6.5.1 Abundances of all taxa in specific sample

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

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

6.5.2 Abundances of specific taxa in all samples

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

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

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

locale:
```

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

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

other attached packages:
 [1] mia_1.3.19                     MultiAssayExperiment_1.20.0
 [3] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
 [5] XVector_0.34.0                 SingleCellExperiment_1.16.0
 [7] SummarizedExperiment_1.24.0    Biobase_2.54.0
 [9] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[11] IRanges_2.28.0                 S4Vectors_0.32.4
[13] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[15] matrixStats_0.62.0-9000        BiocStyle_2.22.0
[17] rebook_1.4.0

loaded via a namespace (and not attached):
 [1] ggbeeswarm_0.6.0            colorspace_2.0-3
 [3] ellipsis_0.3.2              scuttle_1.4.0
 [5] BiocNeighbors_1.12.0        ggrepel_0.9.1
 [7] bit64_4.0.5                 fansi_1.0.3
 [9] decontam_1.14.0             splines_4.1.3
[11] codetools_0.2-18            sparseMatrixStats_1.6.0
[13] cachem_1.0.6                knitr_1.38
[15] scater_1.22.0               jsonlite_1.8.0
[17] cluster_2.1.3               graph_1.72.0
[19] BiocManager_1.30.16         compiler_4.1.3
[21] assertthat_0.2.1            Matrix_1.4-1
[23] fastmap_1.1.0               lazyeval_0.2.2
[25] cli_3.2.0                   BiocSingular_1.10.0
[27] htmltools_0.5.2             tools_4.1.3
[29] rsvd_1.0.5                  gtable_0.3.0
[31] glue_1.6.2                  GenomeInfoDbData_1.2.7
[33] reshape2_1.4.4              dplyr_1.0.8
[35] Rcpp_1.0.8.3                vctrs_0.4.1
[37] ape_5.6-2                   nlme_3.1-157
[39] DECIPHER_2.22.0             DelayedMatrixStats_1.16.0
[41] xfun_0.30                   stringr_1.4.0
[43] beachmat_2.10.0             lifecycle_1.0.1
[45] irlba_2.3.5                 XML_3.99-0.9
```

```
[47] zlibbioc_1.40.0            MASS_7.3-56
[49] scales_1.2.0               parallel_4.1.3
[51] yaml_2.3.5                 memoise_2.0.1
[53] gridExtra_2.3              ggplot2_3.3.5
[55] yulab.utils_0.0.4          stringi_1.7.6
[57] RSQLite_2.2.12             ScaledMatrix_1.2.0
[59] tidytree_0.3.9             permute_0.9-7
[61] filelock_1.0.2             BiocParallel_1.28.3
[63] rlang_1.0.2                pkgconfig_2.0.3
[65] bitops_1.0-7               evaluate_0.15
[67] lattice_0.20-45            purrr_0.3.4
[69] treeio_1.18.1              CodeDepends_0.6.5
[71] bit_4.0.4                  tidyselect_1.1.2
[73] plyr_1.8.7                 magrittr_2.0.3
[75] bookdown_0.26              R6_2.5.1
[77] generics_0.1.2             DelayedArray_0.20.0
[79] DBI_1.1.2                  mgcv_1.8-40
[81] pillar_1.7.0               RCurl_1.98-1.6
[83] tibble_3.1.6               dir.expiry_1.2.0
[85] crayon_1.5.1               utf8_1.2.2
[87] rmarkdown_2.13             viridis_0.6.2
[89] grid_4.1.3                 blob_1.2.3
[91] vegan_2.6-2                digest_0.6.29
[93] tidyr_1.2.0                munsell_0.5.0
[95] DirichletMultinomial_1.36.0 beeswarm_0.4.0
[97] viridisLite_0.4.0          vipor_0.4.5
```

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,
                             abund_values = "counts",
                             index = "observed",
                             name="observed")

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

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

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

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

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

7.1.2 Diversity

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

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

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

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

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

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

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

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

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

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

7.1.3 Faith phylogenetic diversity

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

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

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

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

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

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

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

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

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

7.1.4 Evenness

Evenness can be calculated with `estimateEvenness`.

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

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

7.1.5 Dominance

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

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

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

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

7.1.6 Rarity

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

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

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

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

7.1.7 Divergence

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

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

7.2 Visualization

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

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

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

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

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

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

other attached packages:
 [1] patchwork_1.1.1                ggsignif_0.6.3
 [3] scater_1.22.0                  ggplot2_3.3.5
 [5] scuttle_1.4.0                  mia_1.3.19
 [7] MultiAssayExperiment_1.20.0    TreeSummarizedExperiment_2.1.4
 [9] Biostrings_2.62.0              XVector_0.34.0
[11] SingleCellExperiment_1.16.0    SummarizedExperiment_1.24.0
[13] Biobase_2.54.0                 GenomicRanges_1.46.1
[15] GenomeInfoDb_1.30.1            IRanges_2.28.0
[17] S4Vectors_0.32.4               BiocGenerics_0.40.0
[19] MatrixGenerics_1.6.0           matrixStats_0.62.0-9000
[21] BiocStyle_2.22.0               rebook_1.4.0

loaded via a namespace (and not attached):
  [1] ggbeeswarm_0.6.0            colorspace_2.0-3
  [3] ellipsis_0.3.2              BiocNeighbors_1.12.0
  [5] farver_2.1.0                ggrepel_0.9.1
  [7] bit64_4.0.5                 fansi_1.0.3
  [9] decontam_1.14.0             splines_4.1.3
 [11] codetools_0.2-18            sparseMatrixStats_1.6.0
 [13] cachem_1.0.6                knitr_1.38
 [15] jsonlite_1.8.0              cluster_2.1.3
```

```
[17] graph_1.72.0                BiocManager_1.30.16
[19] compiler_4.1.3              assertthat_0.2.1
[21] Matrix_1.4-1                fastmap_1.1.0
[23] lazyeval_0.2.2              cli_3.2.0
[25] BiocSingular_1.10.0         htmltools_0.5.2
[27] tools_4.1.3                 rsvd_1.0.5
[29] gtable_0.3.0                glue_1.6.2
[31] GenomeInfoDbData_1.2.7      reshape2_1.4.4
[33] dplyr_1.0.8                 Rcpp_1.0.8.3
[35] vctrs_0.4.1                 ape_5.6-2
[37] nlme_3.1-157                DECIPHER_2.22.0
[39] DelayedMatrixStats_1.16.0   xfun_0.30
[41] stringr_1.4.0               beachmat_2.10.0
[43] lifecycle_1.0.1             irlba_2.3.5
[45] XML_3.99-0.9                zlibbioc_1.40.0
[47] MASS_7.3-56                 scales_1.2.0
[49] parallel_4.1.3              yaml_2.3.5
[51] memoise_2.0.1               gridExtra_2.3
[53] yulab.utils_0.0.4           stringi_1.7.6
[55] RSQLite_2.2.12              highr_0.9
[57] ScaledMatrix_1.2.0          tidytree_0.3.9
[59] permute_0.9-7               filelock_1.0.2
[61] BiocParallel_1.28.3         rlang_1.0.2
[63] pkgconfig_2.0.3             bitops_1.0-7
[65] evaluate_0.15               lattice_0.20-45
[67] purrr_0.3.4                 labeling_0.4.2
[69] treeio_1.18.1               CodeDepends_0.6.5
[71] cowplot_1.1.1               bit_4.0.4
[73] tidyselect_1.1.2            plyr_1.8.7
[75] magrittr_2.0.3              bookdown_0.26
[77] R6_2.5.1                    generics_0.1.2
[79] DelayedArray_0.20.0         DBI_1.1.2
[81] withr_2.5.0                 mgcv_1.8-40
[83] pillar_1.7.0                RCurl_1.98-1.6
[85] tibble_3.1.6                dir.expiry_1.2.0
[87] crayon_1.5.1                utf8_1.2.2
[89] rmarkdown_2.13              viridis_0.6.2
[91] grid_4.1.3                  blob_1.2.3
[93] vegan_2.6-2                 digest_0.6.29
[95] tidyr_1.2.0                 munsell_0.5.0
[97] DirichletMultinomial_1.36.0 beeswarm_0.4.0
[99] viridisLite_0.4.0           vipor_0.4.5
```

Chapter 8

Community similarity

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

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

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

8.1 Explained variance

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8.2 Community comparisons by beta diversity analysis

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

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

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

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

```
library(scater)
tse <- runMDS(tse, FUN = vegan::vegdist, method = "bray", name =
 ↪  "PCoA_BC", exprs_values = "counts")
```

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

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

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

print(p)
```

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

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

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

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

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

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

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

library(patchwork)
```

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

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

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

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

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

8.3 Other ordination methods

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

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

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

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

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

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

FIXME: let us switch to UMAP for the examples?

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

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

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

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

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

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

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

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

# Get assay
assay <- t(assay(enterotype, "relabundance"))
# Get colData
coldata <- colData(enterotype)

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

# # Perform RDA
rda <- rda(formula, data = coldata, scale = TRUE, na.action =
 ↪  na.exclude)

# Initialize list for p-values
rda_info <- list()
# Name for storing the result
variable_name <- "all"
# Calculate and store p-value, and other information
rda_info[[variable_name]] <- c(constrained = rda$CCA$tot.chi,
                               unconstrainded = rda$CA$tot.chi,
                               proportion =
                                ↪  rda$CCA$tot.chi/rda$CA$tot.chi,
                                ↪
                               p_value = anova.cca(rda)["Model",
                                ↪  "Pr(>F)"] )

# Loop through variables
permutations <- 999
for( variable_name in variable_names ){
    # Create a formula
    formula <- as.formula(paste0("assay ~ ", variable_name) )
    # Perform RDA
    rda_temp <- rda(formula, data = coldata, scale = TRUE,
 ↪  na.action = na.exclude)
    # Add Info to list
    rda_info[[variable_name]] <- c(constrained =
 ↪  rda_temp$CCA$tot.chi,
```

```
                                    unconstrainded =
                                     ↪  rda_temp$CA$tot.chi,
                                    proportion =
                                     ↪  rda_temp$CCA$tot.chi/rda$CA$tot.chi,
                                     ↪
                                    p_value = anova.cca(rda_temp,
                                     ↪  permutations =
                                     ↪  permutations
                                                        )["Model",
 ↪  "Pr(>F)"] )
}
# Convert into data.frame
rda_info <- t(as.data.frame(rda_info))
rda_info_clean <- rda_info
# Adjust names
colnames(rda_info_clean) <-
    c("Explained by variables", "Unexplained by variables",
     ↪  "Proportion expl by vars",
      paste0("P-value (PERMANOVA ", permutations, "
       ↪  permutations)") )
# Print info
kable(rda_info_clean)
```

	Explained by variables	Unexplained by variables	Proportion expl by vars	P-val
all	35.30	191.7	0.1842	
ClinicalStatus	19.08	209.9	0.0996	
Gender	5.31	223.7	0.0277	
Age	10.59	216.4	0.0552	

```
# Load ggord for plotting
if(!require("ggord")){
    if(!require("devtools")){
        install.packages("devtools")
        library("devtools")
    }
    install_github("https://github.com/fawda123/ggord/")
    library("ggord")
}
if(!require("ggplot2")){
    install.packages("ggplot2")
    library("ggplot2")
}
# Since na.exclude was used, if there were rows missing
 ↪  information, they were
# dropped off. Subset coldata so that it matches with rda.
```

```
coldata <- coldata[ rownames(rda$CCA$wa), ]

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

# Loop through vector labels
vec_lab <- sapply(vec_lab_old, FUN = function(name){
    # Get the variable name
    variable_name <- variable_names[ str_detect(name,
 ↪  variable_names) ]
    # If the vector label includes also group name
    if( !any(name %in% variable_names) ){
        # Get the group names
        group_name <- unique( coldata[[variable_name]] )[
        which( paste0(variable_name, unique(
         ↪  coldata[[variable_name]] )) == name ) ]
        # Modify vector so that group is separated from variable
         ↪  name
        new_name <- paste0(variable_name, " \U2012 ", group_name)
    } else{
        new_name <- name
    }
    # Add percentage how much this variable explains, and p-value
    new_name <- expr(paste(!!new_name, " (",
                           !!format(round(
                            ↪  rda_info[variable_name,
                            ↪  "proportion"]*100, 1), nsmall =
                            ↪  1),
                           "%, ",italic("P"), " = ",
                           !!gsub("0\\.","\\.", format(round(
                            ↪  rda_info[variable_name,
                            ↪  "p_value"], 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 plto, we can see that only age has significant affect on microbial profile.

8.4 Visualizing the most dominant genus on PCoA

In this section we visualize most dominant genus on PCoA. A similar visualization was proposed in Taxonomic signatures of cause-specific mortality risk in human gut microbiome, Salosensaari et al. (2021).

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

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

Performing PCoA with Bray-Curtis dissimilarity.

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

Getting top taxa and visualizing the abundance on PCoA.

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

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

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

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

# Retrieving the explained variance
```

```
e <- attr(reducedDim(tse_Genus, "PCoA_BC"), "eig");
var_explained <- e/sum(e[e>0])*100

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

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

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

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

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

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

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

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

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

8.4.1 Testing differences in community composition between sample groups

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

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

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

We can get equal result by first performing distance-based redundancy analysis (dbRDA), and then applying permutational test for result of redundancy analysis. Advantage is that by doing so we can get coefficients: how much each taxa affect to the result.

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

# Set seed for reproducibility
set.seed(1576)
# We choose 99 random permutations. Consider applying more (999
 ↪  or 9999) in your
# analysis.
permanova <- adonis2(t(assay(tse,"relabundance")) ~ Group,
                     data = colData(tse),
                     method = "euclidean",
                     by = NULL,
                     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,
                        method = "euclidean",
                        permutations = 99)
```

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

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

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

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

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

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

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

8.4.2 Checking the homogeneity condition

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

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

```
## Analysis of Variance Table
##
## Response: Distances
##           Df Sum Sq Mean Sq F value  Pr(>F)
## Groups     1 0.2385  0.2385     103 3.6e-10 ***
## Residuals 24 0.0554  0.0023
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

8.5 Further reading

- How to extract information from clusters

- Chapter 10 on community typing

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

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

other attached packages:
 [1] ggord_1.1.7                    knitr_1.38
 [3] stringr_1.4.0                  patchwork_1.1.1
 [5] scater_1.22.0                  scuttle_1.4.0
 [7] ggplot2_3.3.5                  vegan_2.6-2
 [9] lattice_0.20-45                permute_0.9-7
[11] mia_1.3.19                     MultiAssayExperiment_1.20.0
[13] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
[15] XVector_0.34.0                 SingleCellExperiment_1.16.0
[17] SummarizedExperiment_1.24.0    Biobase_2.54.0
[19] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[21] IRanges_2.28.0                 S4Vectors_0.32.4
[23] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[25] matrixStats_0.62.0-9000        BiocStyle_2.22.0
[27] rebook_1.4.0

loaded via a namespace (and not attached):
 [1] Rtsne_0.16                  ggbeeswarm_0.6.0
 [3] colorspace_2.0-3            ellipsis_0.3.2
```

```
 [5] BiocNeighbors_1.12.0        farver_2.1.0
 [7] ggrepel_0.9.1               bit64_4.0.5
 [9] fansi_1.0.3                 decontam_1.14.0
[11] splines_4.1.3               codetools_0.2-18
[13] sparseMatrixStats_1.6.0     cachem_1.0.6
[15] jsonlite_1.8.0              cluster_2.1.3
[17] graph_1.72.0                BiocManager_1.30.16
[19] compiler_4.1.3              assertthat_0.2.1
[21] Matrix_1.4-1                fastmap_1.1.0
[23] lazyeval_0.2.2              cli_3.2.0
[25] BiocSingular_1.10.0         htmltools_0.5.2
[27] tools_4.1.3                 rsvd_1.0.5
[29] gtable_0.3.0                glue_1.6.2
[31] GenomeInfoDbData_1.2.7      reshape2_1.4.4
[33] dplyr_1.0.8                 Rcpp_1.0.8.3
[35] vctrs_0.4.1                 ape_5.6-2
[37] nlme_3.1-157                DECIPHER_2.22.0
[39] DelayedMatrixStats_1.16.0   xfun_0.30
[41] beachmat_2.10.0             lifecycle_1.0.1
[43] irlba_2.3.5                 XML_3.99-0.9
[45] zlibbioc_1.40.0             MASS_7.3-56
[47] scales_1.2.0                parallel_4.1.3
[49] yaml_2.3.5                  memoise_2.0.1
[51] gridExtra_2.3               yulab.utils_0.0.4
[53] stringi_1.7.6               RSQLite_2.2.12
[55] highr_0.9                   ScaledMatrix_1.2.0
[57] tidytree_0.3.9              filelock_1.0.2
[59] BiocParallel_1.28.3         rlang_1.0.2
[61] pkgconfig_2.0.3             bitops_1.0-7
[63] evaluate_0.15               purrr_0.3.4
[65] labeling_0.4.2              treeio_1.18.1
[67] CodeDepends_0.6.5           cowplot_1.1.1
[69] bit_4.0.4                   tidyselect_1.1.2
[71] plyr_1.8.7                  magrittr_2.0.3
[73] bookdown_0.26               R6_2.5.1
[75] generics_0.1.2              DelayedArray_0.20.0
[77] DBI_1.1.2                   withr_2.5.0
[79] mgcv_1.8-40                 pillar_1.7.0
[81] RCurl_1.98-1.6              tibble_3.1.6
[83] dir.expiry_1.2.0            crayon_1.5.1
[85] utf8_1.2.2                  rmarkdown_2.13
[87] viridis_0.6.2               grid_4.1.3
[89] blob_1.2.3                  digest_0.6.29
[91] tidyr_1.2.0                 munsell_0.5.0
[93] DirichletMultinomial_1.36.0 beeswarm_0.4.0
[95] viridisLite_0.4.0           vipor_0.4.5
```

Chapter 9

Community composition

```
## Loading required package: ecodist
```

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

9.1 Visualizing taxonomic composition

9.1.1 Composition barplot

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

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

# Getting top taxa on a Phylum level
tse_phylum <- agglomerateByRank(tse, rank ="Phylum",
 ↪  onRankOnly=TRUE)
top_taxa <- getTopTaxa(tse_phylum,top = 5, abund_values =
 ↪  "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, abund_values="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 <- transformSamples(tse_phylum, method = "clr",
 ↪  pseudocount = 1)
# Add z-transformation on features (taxa)
tse_phylum <- transformFeatures(tse_phylum, abund_values = "clr",
 ↪
                                 method = "z", name = "clr_z")
# Melts the assay
df <- meltAssay(tse_phylum, abund_values = "clr_z")

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

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

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

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

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

# Does clr-transformation
tse_phylum_subset <- transformSamples(tse_phylum_subset, method =
 ↪  "clr", pseudocount = 1)
# Does z-transformation
tse_phylum_subset <- transformFeatures(tse_phylum_subset,
 ↪  abund_values = "clr",
                                        method = "z", name =
                                         ↪  "clr_z")

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

# Gets the assay table
```

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

# Creates the heatmap
pheatmap(mat)
```

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

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

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

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

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

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

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

taxa_tree
```

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

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

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

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

# Prints taxa and their clusters
taxa_clusters
```

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

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

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

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

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

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

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

```
# Get order of samples in plot
samples_ordered <- rev(get_taxa_name(sample_tree))

sample_tree
```

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

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

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

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

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

sample_data
```

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

Now we can create heatmap with additional annotations.

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

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

In addition to *pheatmap* package, there are also other packages that provide functions for more complex heatmaps, such as *iheatmapr* and ComplexHeatmap.

sechm package provides wrapper for *ComplexHeatmap*.

```
if(!require(sechm)){
  BiocManager::install("sechm")
  library(sechm)
}
# 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,
      genes = 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 similar heatmap by just using *ggplot2*.

```
# 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")

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")

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")

sample_clusters_annotation
```

```
if(!require(reshape2)){
    install.packages("reshape2")
    library(reshape2)
}
# 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
```

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

# Create layout
design <- c(
  area(3, 1, 4, 1),
  area(1, 2, 1, 3),
  area(2, 2, 2, 3),
  area(3, 2, 4, 3)
```

```
)

plot(design)
```

```
# Combine plots
plot <- row_annotation + sample_clusters_annotation +
 ↪  sample_types_annotation + heatmap  +
    plot_layout(design = design, guides = "collect", # Specify
     ↪  layout, collect legends

                # Adjust widths and heights to align plots.
                # When annotation plot is larger, it might not
                 ↪  fit into its column/row.
                # Then you need to make column/row larger.

                # Relative widths and heights of each column and
                 ↪  row:
                # Currently, the width of the first column is 15
                 ↪  % and the height of
                # first two rows are 30 % the size of others

                # To get this work most of the times, you can
                 ↪  adjust all sizes to be 1, i.e. equal,
                # but then the gaps between plots are larger.
```

```
                 widths = c(0.15, 1, 1),
                 heights = c(0.3, 0.3, 1, 1))

plot
```

```
# Create layout
design <- c(
  area(4, 1, 5, 1),
  area(4, 2, 5, 2),
  area(1, 3, 1, 4),
  area(2, 3, 2, 4),
  area(3, 3, 3, 4),
  area(4, 3, 5, 4)
)

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
```

Clusters
Sample types
SAR406
Euryarchaeota
Firmicutes
Tenericutes
Verrucomicrobia
Bacteroidetes
Lentisphaerae
Synergistetes
Proteobacteria
Cyanobacteria
SR1
Fusobacteria
Spirochaetes
Acidobacteria
Thermi
Gemmatimonadetes
Planctomycetes
Crenarchaeota
Actinobacteria
Chloroflexi
Clusters
M11Plmr
M31Plmr
F21Plmr
M31Fcsw
M11Fcsw
TS28
TS29
M31Tong
M11Tong
Clusters
1
2
3
Sample types
Feces
Skin
Tongue
CLR + Z transform
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0

Chapter 10

Community typing

10.1 Dirichlet Multinomial Mixtures (DMM)

This section focus on DMM analysis.

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

Let's cluster the data with DMM clustering.

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

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

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

```
## class: TreeSummarizedExperiment
```

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

Return information on metadata that the object contains.

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

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

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

```
getDMN(tse_dmn)
```

```
## [[1]]
## class: DMN
## k: 1
## samples x taxa: 26 x 67
## Laplace: 7715 BIC: 7802 AIC: 7760
##
## [[2]]
## class: DMN
## k: 2
## samples x taxa: 26 x 67
## Laplace: 7681 BIC: 7902 AIC: 7818
##
## [[3]]
## class: DMN
## k: 3
## samples x taxa: 26 x 67
## Laplace: 7690 BIC: 8076 AIC: 7948
```

```
##
## [[4]]
## class: DMN
## k: 4
## samples x taxa: 26 x 67
## Laplace: 7790 BIC: 8354 AIC: 8183
##
## [[5]]
## class: DMN
## k: 5
## samples x taxa: 26 x 67
## Laplace: 7849 BIC: 8554 AIC: 8341
##
## [[6]]
## class: DMN
## k: 6
## samples x taxa: 26 x 67
## Laplace: 7927 BIC: 8797 AIC: 8541
##
## [[7]]
## class: DMN
## k: 7
## samples x taxa: 26 x 67
## Laplace: 7997 BIC: 9014 AIC: 8715
```

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

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

Return the model that has the best fit.

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

```
## class: DMN
## k: 2
## samples x taxa: 26 x 67
## Laplace: 7681 BIC: 7902 AIC: 7818
```

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

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

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

dmn_group
```

```
## class: DMNGroup
```

```
## summary:
##                     k samples taxa    NLE  LogDet Laplace    BIC  AIC
## Feces               2       4   67 1078.3 -106.14   901.2 1171.9 1213
## Freshwater          2       2   67  889.6  -97.28   716.9  936.4 1025
## Freshwater (creek) 2       3   67 1600.3  787.33  1869.9 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(getBestDMNFit(tse_dmn))
```

```
##       pi  theta
## 1 0.8846  10.76
## 2 0.1154 349.00
```

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

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

```
##         [,1]       [,2]
## CL3        1  0.000e+00
## CC1        1  0.000e+00
## SV1        1 2.938e-281
## M31Fcsw    1  0.000e+00
## M11Fcsw    1 3.902e-299
## M31Plmr    1  0.000e+00
```

Contribution of each taxa to each component

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

```
##                           [,1]    [,2]
## Phylum:Crenarchaeota  0.167446  1.4648
## Phylum:Euryarchaeota  0.155766  2.4076
## Phylum:Actinobacteria 0.882684  6.1715
## Phylum:Spirochaetes   0.144895  1.0871
## Phylum:MVP-15         0.004283  0.2159
## Phylum:Proteobacteria 2.300469 62.3813
```

Get the assignment probabilities

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

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

Computing the euclidean PCoA and storing it as a data frame

```
# Does clr transformation. Pseudocount is added, because data
 ↪  contains zeros.
tse <- transformCounts(tse, method = "clr", pseudocount = 1)

# Gets clr table
clr_assay <- assays(tse)$clr

# Transposes it to get taxa to columns
clr_assay <- t(clr_assay)

# Calculates Euclidean distances between samples. Because taxa is
 ↪  in columns,
# it is used to compare different samples.
euclidean_dist <- vegan::vegdist(clr_assay, method = "euclidean")

# Does principal coordinate analysis
euclidean_pcoa <- ecodist::pco(euclidean_dist)

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

```
# Creates a data frame that contains principal coordinates and
 ↪  DMM information
euclidean_dmm_pcoa_df <- cbind(euclidean_pcoa_df,
                               dmm_component = vec)
# Creates a plot
euclidean_dmm_plot <- ggplot(data = euclidean_dmm_pcoa_df,
                             aes(x=pcoa1, y=pcoa2,
```

```
                                  color = dmm_component)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "PCoA with Aitchison distances") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_dmm_plot
```

10.2 Community Detection

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

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

Installing package:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

k = 3
0.2
0.1
0.0
−0.1
1 2 3 4 5

k = 5
0.0
0.2
0.4
0.6
1
2
3

k = 10
0.0
0.2
0.4
0.6
1
2
3
4

10.3 Additional Community Typing

For more community typing techniques applied to the 'SprockettTHData' data set, see the attached .Rmd file.

Link:

- Rmd

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

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

other attached packages:
 [1] scater_1.22.0                  scuttle_1.4.0
 [3] bluster_1.4.0                  patchwork_1.1.1
 [5] reshape2_1.4.4                 sechm_1.2.0
 [7] ggtree_3.2.1                   ape_5.6-2
 [9] pheatmap_1.0.12                miaViz_1.3.3
[11] ggraph_2.0.5                   ggplot2_3.3.5
[13] mia_1.3.19                     MultiAssayExperiment_1.20.0
[15] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
[17] XVector_0.34.0                 SingleCellExperiment_1.16.0
[19] SummarizedExperiment_1.24.0    Biobase_2.54.0
[21] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
```

```
[23] IRanges_2.28.0                S4Vectors_0.32.4
[25] BiocGenerics_0.40.0           MatrixGenerics_1.6.0
[27] matrixStats_0.62.0-9000       ecodist_2.0.7
[29] BiocStyle_2.22.0              rebook_1.4.0

loaded via a namespace (and not attached):
  [1] circlize_0.4.14            plyr_1.8.7
  [3] igraph_1.3.1               lazyeval_0.2.2
  [5] splines_4.1.3              BiocParallel_1.28.3
  [7] digest_0.6.29              foreach_1.5.2
  [9] yulab.utils_0.0.4          htmltools_0.5.2
 [11] viridis_0.6.2              fansi_1.0.3
 [13] magrittr_2.0.3             memoise_2.0.1
 [15] ScaledMatrix_1.2.0         doParallel_1.0.17
 [17] cluster_2.1.3              DECIPHER_2.22.0
 [19] ComplexHeatmap_2.10.0      graphlayouts_0.8.0
 [21] colorspace_2.0-3           blob_1.2.3
 [23] ggrepel_0.9.1              xfun_0.30
 [25] dplyr_1.0.8                crayon_1.5.1
 [27] RCurl_1.98-1.6             jsonlite_1.8.0
 [29] graph_1.72.0               iterators_1.0.14
 [31] glue_1.6.2                 polyclip_1.10-0
 [33] registry_0.5-1             gtable_0.3.0
 [35] zlibbioc_1.40.0            V8_4.1.0
 [37] GetoptLong_1.0.5           DelayedArray_0.20.0
 [39] BiocSingular_1.10.0        shape_1.4.6
 [41] scales_1.2.0               DBI_1.1.2
 [43] randomcoloR_1.1.0.1        Rcpp_1.0.8.3
 [45] viridisLite_0.4.0          clue_0.3-60
 [47] decontam_1.14.0            gridGraphics_0.5-1
 [49] tidytree_0.3.9             bit_4.0.4
 [51] rsvd_1.0.5                 FNN_1.1.3
 [53] RColorBrewer_1.1-3         dir.expiry_1.2.0
 [55] ellipsis_0.3.2             pkgconfig_2.0.3
 [57] XML_3.99-0.9               farver_2.1.0
 [59] uwot_0.1.11                CodeDepends_0.6.5
 [61] utf8_1.2.2                 ggplotify_0.1.0
 [63] tidyselect_1.1.2           labeling_0.4.2
 [65] rlang_1.0.2                munsell_0.5.0
 [67] tools_4.1.3                cachem_1.0.6
 [69] cli_3.2.0                  DirichletMultinomial_1.36.0
 [71] generics_0.1.2             RSQLite_2.2.12
 [73] evaluate_0.15              stringr_1.4.0
 [75] fastmap_1.1.0              yaml_2.3.5
 [77] knitr_1.38                 bit64_4.0.5
 [79] tidygraph_1.2.1            purrr_0.3.4
```

```
 [81] nlme_3.1-157                sparseMatrixStats_1.6.0
 [83] aplot_0.1.3                 compiler_4.1.3
 [85] curl_4.3.2                  png_0.1-7
 [87] beeswarm_0.4.0              filelock_1.0.2
 [89] treeio_1.18.1               tibble_3.1.6
 [91] tweenr_1.0.2                stringi_1.7.6
 [93] highr_0.9                   lattice_0.20-45
 [95] Matrix_1.4-1                vegan_2.6-2
 [97] permute_0.9-7               vctrs_0.4.1
 [99] pillar_1.7.0                lifecycle_1.0.1
[101] BiocManager_1.30.16         GlobalOptions_0.1.2
[103] BiocNeighbors_1.12.0        cowplot_1.1.1
[105] bitops_1.0-7                irlba_2.3.5
[107] seriation_1.3.5             R6_2.5.1
[109] TSP_1.2-0                   bookdown_0.26
[111] gridExtra_2.3               vipor_0.4.5
[113] codetools_0.2-18            MASS_7.3-56
[115] assertthat_0.2.1            rjson_0.2.21
[117] withr_2.5.0                 GenomeInfoDbData_1.2.7
[119] mgcv_1.8-40                 parallel_4.1.3
[121] grid_4.1.3                  ggfun_0.0.6
[123] beachmat_2.10.0             tidyr_1.2.0
[125] rmarkdown_2.13              DelayedMatrixStats_1.16.0
[127] Rtsne_0.16                  ggnewscale_0.4.7
[129] ggforce_0.3.3               ggbeeswarm_0.6.0
```

Chapter 11

Biclustering

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

Here, we use following packages: - biclust - cobiclust

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

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

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

Check more ideas for heatmaps from here.

11.1 Taxa vs samples

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

Load example data

```
library(mia)
if(!require(microbiomeDataSets)){
    BiocManager::install("microbiomeDataSets")
    library(microbiomeDataSets)
```

```
}

mae <- HintikkaXOData()
```

Only the most prevalent taxa are included in analysis.

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

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

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

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

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

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

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

# Print clusters
clusters$classification
```

```
## $rowclass
##  [1] 1 1 1 1 2 2 1 1 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 2 1 1 2 1 1 1 1 1 2 1 1 2 1 1
## [39] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1
##
```

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

Next we can plot clusters. Commonly used plot is heatmap with annotations.

```r
if(!require(pheatmap)){
    install.packages("pheatmap")
    library(pheatmap)
}
# z-transform for heatmap
mae[[1]] <- transformFeatures(mae[[1]], abund_values = "clr",
 ↪  method = "z", name = "clr_z")

# Create annotations. When column names are equal, they should
 ↪  share levels.
# Here samples include 3 clusters, and taxa 2. That is why we
 ↪  have to make
# column names unique.
annotation_col <- data.frame(colData(mae[[1]])[, "clusters", drop
 ↪  = F])
colnames(annotation_col) <- "col_clusters"

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

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

Boxplot is commonly used to summarize the results:

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

# ggplot requires data in melted format
melt_assay <- meltAssay(mae[[1]], abund_values = "clr",
 ↪  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 = clr)) +
  labs(x = "Taxa clusters")
```

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

p1 + p2
```

11.2 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.

```
############################ REMOVE THIS WHEN BUG IN
 ↪  GETEXPERIMENTCROSSCORR IS FIXED
rownames(mae[[1]]) <- make.unique(rownames(mae[[1]]))

# Calculate correlations
corr <- getExperimentCrossCorrelation(mae, 1, 2,
                                      abund_values1 = "clr",
                                      abund_values2 = "nmr",
                                      mode = "matrix",
                                      cor_threshold = 0.2)
```

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

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

# Set seed for reproducibility
set.seed(3973)

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

bc
```

```
##
## An object of class Biclust
##
## call:
##  biclust(x = corr, method = BCPlaid(), fit.model = y ~ m, background = TRUE,
##      shuffle = 100, back.fit = 0, max.layers = 10, iter.startup = 10,
##      iter.layer = 100, verbose = FALSE)
##
## There was no cluster found
```

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

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

```
# Functions for obtaining biclust information

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

```
  bc_rows <- bc@RowxNumber
  bc_rows <- data.frame(bc_rows)

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

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

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

  # If no clusters were found, create one. Otherwise create
   ↪  additional cluster which
  # contain those samples that are not included in clusters that
   ↪  were found.
  if( nrow(bc_clusters) != dim ){
      bc_clusters <- data.frame(cluster = rep(TRUE, dim))
  } else {
      # Create additional cluster that includes those
       ↪  samples/features that
      # are not included in other clusters.
      vec <- ifelse(rowSums(bc_clusters) > 0, FALSE, TRUE)
      # If additional cluster contains samples, then add it
      if ( any(vec) ){
          bc_clusters <- cbind(bc_clusters, vec)
      }
  }
  # Adjust row and column names
  rownames(bc_clusters) <- names
  colnames(bc_clusters) <- paste0("cluster_",
   ↪  1:ncol(bc_clusters))
  return(bc_clusters)
}
```

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

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

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

```
##                            cluster_1
## D_5__Ruminiclostridium 5       TRUE
## D_5__Lactobacillus             TRUE
## D_5__uncultured                TRUE
## D_5__uncultured bacterium      TRUE
## D_5__Lactococcus               TRUE
## D_5__Lachnoclostridium         TRUE
```

Let's collect information for the scatter plot.

```r
# Function for obtaining sample-wise sum, mean, median, and mean
↪  variance for each cluster
.sum_mean_median_var <- function(tse1, tse2, abund_values1,
↪  abund_values2, 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, abund_values1)
    assay2 <- assay(tse_subset2, abund_values2)
    # 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]], "clr", "nmr",
 ↪  bicluster_rows, bicluster_columns)
```

Now we can create a scatter plot. X-axis includes median clr abundance of microbiome and y-axis median absolute concentration of each metabolite. Each data point represents a single sample.

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

```
pics <- list()
for(i in seq_along(df)){
  pics[[i]] <- ggplot(df[[i]])  +
      geom_point(aes(x = median1, y = median2)) +
      labs(title = paste0("Cluster ", i),
           x = "Taxa (clr 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)
```

11.3 Taxa vs taxa

Third and final example deals with situation where we want to analyze correlation between taxa. *biclust* is suitable for this.

```
# Calculate cross-correlation
corr <- getExperimentCrossCorrelation(mae, 1, 1,
                                      abund_values1 = "clr",
                                       ↪  abund_values2 = "clr",
                                      mode = "matrix",
                                      cor_threshold = 0.2,
                                       ↪  verbose = F,
                                       ↪  show_warning = F)
```

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

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

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

```
# Create a column that combines information
# If row/column includes in multiple clusters, cluster numbers
 ↪  are separated with "_&_"
bicluster_columns$clusters <- apply(bicluster_columns, 1,

                                     ↪  function(x){paste(paste(which(x)),
                                     ↪  collapse = "_&_") })
bicluster_columns <- bicluster_columns[, "clusters", drop =
 ↪  FALSE]

bicluster_rows$clusters <- apply(bicluster_rows, 1,

                                  ↪  function(x){paste(paste(which(x)),
                                  ↪  collapse = "_&_") })
bicluster_rows <- bicluster_rows[, "clusters", drop = FALSE]
```

```
# Convert boolean values into factor
bicluster_columns <- data.frame(apply(bicluster_columns, 2,
 ↪  as.factor))
bicluster_rows <- data.frame(apply(bicluster_rows, 2, as.factor))

pheatmap(corr, cluster_cols = F, cluster_rows = F,
         annotation_col = bicluster_columns,
         annotation_row = bicluster_rows)
```

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

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

other attached packages:
 [1] biclust_2.0.3                  lattice_0.20-45
 [3] colorspace_2.0-3               MASS_7.3-56
 [5] patchwork_1.1.1                ggplot2_3.3.5
 [7] pheatmap_1.0.12                cobiclust_0.1.0
 [9] microbiomeDataSets_1.1.5       mia_1.3.19
[11] MultiAssayExperiment_1.20.0    TreeSummarizedExperiment_2.1.4
[13] Biostrings_2.62.0              XVector_0.34.0
[15] SingleCellExperiment_1.16.0    SummarizedExperiment_1.24.0
[17] Biobase_2.54.0                 GenomicRanges_1.46.1
[19] GenomeInfoDb_1.30.1            IRanges_2.28.0
[21] S4Vectors_0.32.4               BiocGenerics_0.40.0
[23] MatrixGenerics_1.6.0           matrixStats_0.62.0-9000
[25] BiocStyle_2.22.0               rebook_1.4.0

loaded via a namespace (and not attached):
  [1] AnnotationHub_3.2.2           BiocFileCache_2.2.1
  [3] plyr_1.8.7                    lazyeval_0.2.2
  [5] splines_4.1.3                 BiocParallel_1.28.3
  [7] scater_1.22.0                 digest_0.6.29
  [9] yulab.utils_0.0.4             htmltools_0.5.2
 [11] viridis_0.6.2                 fansi_1.0.3
 [13] magrittr_2.0.3                memoise_2.0.1
 [15] ScaledMatrix_1.2.0            cluster_2.1.3
 [17] DECIPHER_2.22.0               blob_1.2.3
 [19] rappdirs_0.3.3                ggrepel_0.9.1
 [21] xfun_0.30                     dplyr_1.0.8
 [23] crayon_1.5.1                  RCurl_1.98-1.6
 [25] jsonlite_1.8.0                graph_1.72.0
 [27] ape_5.6-2                     glue_1.6.2
 [29] gtable_0.3.0                  zlibbioc_1.40.0
 [31] DelayedArray_0.20.0           additivityTests_1.1-4
 [33] BiocSingular_1.10.0           scales_1.2.0
 [35] DBI_1.1.2                     Rcpp_1.0.8.3
 [37] viridisLite_0.4.0             xtable_1.8-4
 [39] decontam_1.14.0               tidytree_0.3.9
 [41] bit_4.0.4                     rsvd_1.0.5
 [43] httr_1.4.2                    RColorBrewer_1.1-3
 [45] dir.expiry_1.2.0              modeltools_0.2-23
 [47] ellipsis_0.3.2                farver_2.1.0
 [49] pkgconfig_2.0.3               XML_3.99-0.9
```

```
 [51] scuttle_1.4.0                CodeDepends_0.6.5
 [53] dbplyr_2.1.1                 utf8_1.2.2
 [55] labeling_0.4.2               tidyselect_1.1.2
 [57] rlang_1.0.2                  reshape2_1.4.4
 [59] later_1.3.0                  AnnotationDbi_1.56.2
 [61] munsell_0.5.0                BiocVersion_3.14.0
 [63] tools_4.1.3                  cachem_1.0.6
 [65] cli_3.2.0                    DirichletMultinomial_1.36.0
 [67] generics_0.1.2               RSQLite_2.2.12
 [69] ExperimentHub_2.2.1          evaluate_0.15
 [71] stringr_1.4.0                fastmap_1.1.0
 [73] yaml_2.3.5                   knitr_1.38
 [75] bit64_4.0.5                  purrr_0.3.4
 [77] KEGGREST_1.34.0              nlme_3.1-157
 [79] sparseMatrixStats_1.6.0      mime_0.12
 [81] flexclust_1.4-1              compiler_4.1.3
 [83] beeswarm_0.4.0               filelock_1.0.2
 [85] curl_4.3.2                   png_0.1-7
 [87] interactiveDisplayBase_1.32.0 treeio_1.18.1
 [89] tibble_3.1.6                 stringi_1.7.6
 [91] highr_0.9                    Matrix_1.4-1
 [93] vegan_2.6-2                  permute_0.9-7
 [95] vctrs_0.4.1                  pillar_1.7.0
 [97] lifecycle_1.0.1              BiocManager_1.30.16
 [99] BiocNeighbors_1.12.0         bitops_1.0-7
[101] irlba_2.3.5                  httpuv_1.6.5
[103] R6_2.5.1                     bookdown_0.26
[105] promises_1.2.0.1             gridExtra_2.3
[107] vipor_0.4.5                  codetools_0.2-18
[109] assertthat_0.2.1             withr_2.5.0
[111] GenomeInfoDbData_1.2.7       mgcv_1.8-40
[113] parallel_4.1.3               beachmat_2.10.0
[115] class_7.3-20                 tidyr_1.2.0
[117] rmarkdown_2.13               DelayedMatrixStats_1.16.0
[119] shiny_1.7.1                  ggbeeswarm_0.6.0
```

Chapter 12

Differential abundance

12.1 Differential abundance analysis

This section provides an overview and examples of *differential abundance analysis (DAA)* based on one of the openly available datasets in mia to illustrate how to perform differential abundance analysis (DAA). DAA identifies differences in the abundances of individual taxonomic groups between two or more groups (e.g. treatment vs control). This can be performed at any phylogenetic level.

We perform DAA to identify biomarkers and/or gain understanding of a complex system by looking at its isolated components. For example, identifying that a bacterial taxon is different between e.g. a patient group with disease X vs a healthy control group might lead to important insights into the pathophysiology. Changes in the microbiota might be causal or a consequence of the disease. Either way, it can help to understand the system as a whole. Be aware that this approach has also been criticized recently (Quinn et al., 2021).

12.1.1 Examples and tools

There are many tools to perform DAA. The most popular tools, without going into evaluating whether or not they perform well for this task, are:
- ALDEx2
- ANCOM-BC
- corncob
- DESeq2
- edgeR
- LEFse
- MaAsLin2
- metagenomeSeq

- limma voom
- t-test
- Wilcoxon test

We recommend to have a look at Nearing et al. (2021) who compared all these listed methods across 38 different datasets. Because different methods have different approaches (parametric vs non-parametric, different normalization techiniques etc.) to perform the same task (test differential abundance), results can differ between methods. Unfortunately, as Nearing et al. (2021) point out, they differ disturbingly much. Therefore, it is highly recommended to pick several methods to get an idea about how robust and potentially reproducible your findings are depending on the method. In this section we demonstrate 3 methods that can be recommended based on this recent review (ANCOM-BC, ALDEx2 and Maaslin2) and we will compare the results between them. Note that the purpose of this section is to show how to perform DAA in R, not how to correctly do causal inference. E.g. there might be confounding factors that might drive (the absence of) differences between the shown groups that we ignore for simplicity. However, we will show how you could include covariates in those models. Furthermore, we picked a dataset that merely has microbial abundances in a TSE object as well as a grouping variable in the sample data. We simplify the analysis by only including 2 of the 3 groups.

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

# we use the dmn_se dataset and restrict it to
# obese vs lean for easy illustration
data(dmn_se)
se <- dmn_se
# To enable all features and advantages of TreeSE, we convert the
 ↪  object from SE to TreeSE
tse <- as(se, "TreeSummarizedExperiment")
tse <- tse[ ,colData(tse)$pheno != "Overwt"]
colData(tse)$pheno <- fct_drop(colData(tse)$pheno, "Overwt")
# how many observations do we have per group?
count(as.data.frame(colData(tse)), pheno) %>% kable()
```

pheno	n
Lean	61
Obese	193

```
# set a seed because some tools can randomly vary and then
 ↪  produce
# different results:
set.seed(1)
```

12.1.2 Prevalence Filtering

Before we jump to our analyses, we may want to perform prevalence filtering. Nearing et al. (2021) found that applying a 10% threshold for the prevalence of the taxa generally resulted in more robust results. Some tools have builtin arguments for that. By applying the threshold to our input data, we can make sure it is applied for all tools. Below we show how to do this in `mia`:

```
tse <- subsetByPrevalentTaxa(tse, detection = 0, prevalence =
 ↪  0.1)
```

12.1.3 ALDEx2

In this section, we will show how to perform a simple ALDEx2 analysis. If you would choose 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. (2021). Please also have a look at the more extensive vignette that covers this flexible tool in more depth. ALDEx2 estimates technical variation within each sample per taxon by utilizing the Dirichlet distribution. It furthermore applies the centered-log-ratio transformation (or closely related log-ratio transforms). Depending on the experimental setup, it will perform a two sample Welch's T-test and Wilcoxon-test or a one-way ANOVA and Kruskal-Wallis-test. For more complex study designs, there is a possibility to utilize the `glm` functionality within ALDEx2. The Benjamini-Hochberg procedure is applied in any case to correct for multiple testing. Below we show a simple example that illustrates the workflow.

```
# Generate Monte Carlo samples of the Dirichlet distribution for
 ↪  each sample.
# Convert each instance using the centred log-ratio transform.
# This is the input for all further analyses.
x <- aldex.clr(
  reads = assay(tse),
  conds = colData(tse)$pheno,
  # 128 recommened for ttest, 1000 for rigorous effect size
   ↪  calculation
```

```
  mc.samples = 128,
  denom = "all",
  verbose = FALSE
)
# 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)
# 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.

```
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 pvalue. 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 an effect size cutoff of 1 rather than only interpreting the pvalue. Again, this is not the case for any feature.

```
rownames_to_column(aldex_out, "genus") %>%
  filter(wi.eBH <= 0.05)  %>% # here we chose the wilcoxon output
   ↪  rather than tt
  select(genus, we.eBH, wi.eBH, effect, overlap) %>%
  kable()
```

genus	we.eBH	wi.eBH	effect	overlap
Alistipes	0.0009	0.0001	-0.3823	0.2979
Barnesiella	0.0442	0.0066	-0.3229	0.3489
Catenibacterium	0.0266	0.0330	0.2713	0.3718
Lactobacillus	0.0282	0.0183	0.2983	0.3537
Megasphaera	0.0000	0.0001	0.5249	0.2758
Oscillibacter	0.0004	0.0014	-0.3681	0.3291
Parabacteroides	0.0541	0.0133	-0.2832	0.3509
Phascolarctobacterium	0.0238	0.0077	-0.3491	0.3404
Uknown	0.0786	0.0439	-0.2474	0.3852

12.1.4 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., 2021). 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.

As we will see below, to obtain results, all that is needed is to pass a phyloseq object to the `ancombc()` function. Therefore, below we first convert our `TreeSE` object to a `phyloseq` object. Then, we specify the formula. In this formula, other covariates could potentially be included to adjust for confounding. We show this further below. Please check the function documentation to learn about the additional arguments that we specify below.

```
# currently, ancombc requires the phyloseq format, but we can
 ↪  easily convert:
pseq <- makePhyloseqFromTreeSummarizedExperiment(tse)

# perform the analysis
out = ancombc(
  phyloseq = pseq,
  formula = "pheno",
  p_adj_method = "fdr",
  zero_cut = 1, # no prev filtering necessary anymore
```

```
  lib_cut = 0,
  group = "pheno",
  struc_zero = TRUE,
  neg_lb = TRUE,
  tol = 1e-5,
  max_iter = 100,
  conserve = TRUE,
  alpha = 0.05,
  global = TRUE
)
# store the results in res
res <- out$res
```

The object `out` contains all model output. Again, see the documentation of the function under **Value** for an explanation of all the output objects. Our question whether taxa are differentially abundant can be answered by looking at the `res` object, which now contains dataframes with the coefficients, standard errors, p-values and q-values. Conveniently, there is a dataframe `diff_abn`. Here, for each taxon it is indicated whether it is differentially abundant between the groups. Below we show the first 6 entries of this dataframe:

```
kable(head(res$diff_abn))
```

	phenoObese
Acetanaerobacterium	TRUE
Acetivibrio	FALSE
Acidaminococcus	TRUE
Akkermansia	FALSE
Alistipes	TRUE
Allisonella	FALSE

12.1.5 MaAsLin2

Lastly, we will illustrate how to use MaAsLin2, which is the next generation of MaAsLin. As it is based on generalized linear models, it is flexible for different study designs and covariate structures. The official package tutorial can be found here.

```
# maaslin expects features as columns and samples as rows
# for both the asv/otu table as well as meta data
asv <- t(assay(tse))
meta_data <- data.frame(colData(tse))
# you can specifiy different GLMs/normalizations/transforms. We
 ↪  used similar
```

```r
# settings as in Nearing et al. (2021) here:
fit_data <- Maaslin2(
  asv,
  meta_data,
  output = "DAA example",
  transform = "AST",
  fixed_effects = "pheno",
  # 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 = "pheno,Lean",
  normalization = "TSS",
  standardize = FALSE,
  min_prevalence = 0 # prev filterin already done
)
```

```r
# which genera are identified as differentially abundant? (leave
↪  out "head" to
# see all)
kable(head(filter(fit_data$results, qval <= 0.05)))
```

feature	metadata	value	coef	stderr	pval	name	qval	N
Megasphaera	pheno	Obese	0.0489	0.0093	0	phenoObese	0e+00	254
Barnesiella	pheno	Obese	-0.0297	0.0068	0	phenoObese	2e-04	254
Parabacteroides	pheno	Obese	-0.0219	0.0050	0	phenoObese	2e-04	254
Phascolarctobacterium	pheno	Obese	-0.0325	0.0072	0	phenoObese	2e-04	254
Alistipes	pheno	Obese	-0.0523	0.0123	0	phenoObese	3e-04	254
Desulfovibrio	pheno	Obese	-0.0134	0.0032	0	phenoObese	3e-04	254

```r
# A folder will be created that is called like the above
↪  specified output.
# It contains also figures to visualize the difference between
↪  genera
# for the significant ones.
```

12.1.6 Comparison of the methods

When we compare the methods in the context of a research question, we could look at e.g. at whether they agree based on the applied decision criterion (e.g. adjusted p value < 0.05). That is what we illustrate here. First we will look at how many taxa were identified by each method to begin with. In the next step we will look at the intersection of identified taxa. To achieve that, we first create

a dataframe that summarises the decision criterion for each method and shows a score from 0 to 3 indicating how many methods agreed on a particular taxon.

```
summ <- full_join(
    rownames_to_column(aldex_out, "genus") %>%
      select(genus, aldex2 = wi.eBH),
    rownames_to_column(out$res$diff_abn, "genus") %>%
      select(genus, ancombc = phenoObese),
    by = "genus") %>%
  full_join(
    select(fit_data$results, genus = feature, maaslin2 = qval),
    by = "genus") %>%
  mutate(
    across(c(aldex2, maaslin2), ~ .x <= 0.05),
    # the following line would be necessary without prevalence
     ↪  filtering
    # as some methods output NA
    #across(-genus, function(x) ifelse(is.na(x), FALSE, x)),
    score = rowSums(across(c(aldex2, ancombc, maaslin2)))
  )

# This is how it looks like:
kable(head(summ))
```

genus	aldex2	ancombc	maaslin2	score
Acetanaerobacterium	FALSE	TRUE	TRUE	2
Acetivibrio	FALSE	FALSE	FALSE	0
Acidaminococcus	FALSE	TRUE	TRUE	2
Akkermansia	FALSE	FALSE	FALSE	0
Alistipes	TRUE	TRUE	TRUE	3
Allisonella	FALSE	FALSE	FALSE	0

Now we can answer our questions:

```
# how many genera were identified by each method?
summarise(summ, across(where(is.logical), sum)) %>%
  kable()
```

aldex2	ancombc	maaslin2
9	22	16

```
# which genera are identified by all methods?
filter(summ, score == 3) %>% kable()
```

genus	aldex2	ancombc	maaslin2	score
Alistipes	TRUE	TRUE	TRUE	3
Barnesiella	TRUE	TRUE	TRUE	3
Catenibacterium	TRUE	TRUE	TRUE	3
Lactobacillus	TRUE	TRUE	TRUE	3
Megasphaera	TRUE	TRUE	TRUE	3
Oscillibacter	TRUE	TRUE	TRUE	3
Parabacteroides	TRUE	TRUE	TRUE	3
Phascolarctobacterium	TRUE	TRUE	TRUE	3

We see that each method identified at least 9 genera as differentially abundant. Eight of those that were identified by ALDEx2, were also identified by both of the other methods. We could plot the data for any method or for those taxa that were identified by all methods:

```
plot_data <- data.frame(t(assay(tse)))
plot_data$pheno <- colData(tse)$pheno
# create a plot for each genus where the score is indicated in
 ↪  the title
plots <- pmap(select(summ, genus, score), function(genus, score)
 ↪  {
  ggplot(plot_data, aes_string("pheno", genus)) +
    geom_boxplot(aes(fill = pheno), outlier.shape = NA) +
    geom_jitter(width = 0.2, alpha = 0.5) +
    ggtitle(glue::glue("Robustness score {score}")) +
    theme_bw() +
    theme(legend.position = "none")
})

# now we can show only those genera that have at least score 3
 ↪  (or 2 or 1)
robust_plots <- plots[summ$score == 3]


# to display this nicely in the book we use patchwork here:
# (we show first 8)
robust_plots[[1]] +
  robust_plots[[2]] +
  robust_plots[[3]] +
  robust_plots[[4]] +
  robust_plots[[5]] +
  robust_plots[[6]] +
  robust_plots[[7]] +
  robust_plots[[8]] +
  plot_layout(nrow = 2)
```

```
# or if we have most trust in any specific method we can show genera that
# are differentially abundant according to that method and then look in the
# title how many methods also identified it (we only show first 6 here):
ancombc_plots <- plots[summ$ancombc]
ancombc_plots[[1]] +
  ancombc_plots[[2]] +
  ancombc_plots[[3]] +
  ancombc_plots[[4]] +
  ancombc_plots[[5]] +
  ancombc_plots[[6]]
```

12.1.7 Confounding variables

To perform causal inference, it is crucial that the method is able to include covariates in the model. This is not possible with e.g. the Wilcoxon test. Other methods such as both ANCOM methods, ALDEx2, DESeq2, MaAsLin2 and others allow this. Below we show how to include a covariate in ANCOM-BC. It is very similar for all the methods that allow this. Since in this dataset there are no covariates, I first simulate a new variable and add it to the TSE object.

```
# to join new data to existing colData we need to put rownames as
 ↪  a column
colData(tse)$sample_id <- rownames(colData(tse))
# simulate a covariate that I will add to the colData.
df_sim <- tibble(
  sample_id = colData(tse)$sample_id,
  age = rnorm(n = length(colData(tse)$sample_id))
)
# an easy way to join data is to use dplyr functions. The package
 ↪
# tidySummarizedExperiment enables this functionality
tse <- full_join(tse, df_sim, by = "sample_id")
# now the data from df_sim is in the tse object and we can again
 ↪  repeat
# the steps as above:
```

```
pseq <- makePhyloseqFromTreeSummarizedExperiment(tse)
out_cov = ancombc(
  phyloseq = pseq,
  formula = "pheno + age", # here we add age to the model
  p_adj_method = "fdr",
  zero_cut = 0.90,
  lib_cut = 0,
  group = "pheno",
  struc_zero = TRUE,
  neg_lb = TRUE,
  tol = 1e-5,
  max_iter = 100,
  conserve = TRUE,
  alpha = 0.05,
  global = TRUE
)
# now the model answers the question: holding age constant, are
# bacterial taxa differentially abundant? Or, if that is of
 ↪  interest,
# holding phenotype constant, is age associated with bacterial
 ↪  abundance?
# Again we only show the first 6 entries.
kable(head(out_cov$res$diff_abn))
```

	phenoObese	age
Acetanaerobacterium	TRUE	FALSE
Acetivibrio	FALSE	FALSE
Acidaminococcus	TRUE	FALSE
Akkermansia	FALSE	FALSE
Alistipes	TRUE	FALSE
Allisonella	FALSE	FALSE

In the next section of this book chapter we cover methods that can also take into account the phylogenetic information of bacterial taxa to perform group-wise associations.

12.2 Tree-based methods

12.2.1 Group-wise associations testing based on balances with fido

TreeSummarizedExperiment frequently includes a Phylogenetic tree along with associated data about the experiment (at `colData`), that holds covariates which can be used for analyzing group-wise associations.

Such an analysis could be performed with the function `pibble` from the `fido` package, that offers a Multinomial Logistic-Normal Linear Regression model; see vignette of package.

The following presents such an exemplary analysis based on the data of Sprockett et al. (2020) available through `microbiomeDataSets` package.

```
if (!require(fido)){
  # installing the fido package
  devtools::install_github("jsilve24/fido")
}
```

Loading the libraries and importing data:

```
library(fido)
library(mia)
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:

```
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 samples with na values at the covariates (features) under analysis:

```
tse <- tse[ , !is.na(colData(tse)$Delivery_Mode) ]
tse <- tse[ , !is.na(colData(tse)$Age_Years) ]
```

We agglomerate the data at a Phylum rank. Note: Large assay data (along with the covariates/features data) could prevent the analysis later, since the computation will construct matrices that would not always fit memory.

```
tse_phylum <- agglomerateByRank(tse, "Phylum")
```

We extract the counts assay and feature data to build the model matrix having an extra row of ones presenting the intercept for the regression task later:

```
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 the data at `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 predictive distribution:

```
ppc_summary(posterior)
```

```
## Proportions of Observations within 95% Credible Interval: 0.998
```

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])
```

Seemingly the covariate "Age_Years" does not have effect on the model as "Delivery_Mode" would, and "Sex" to some extent. Let's take a closer look at the two latter ones:

```
plot(posterior, par="Lambda", focus.cov = rownames(X)[c(2,4)])
```

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

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

attached base packages:
```

```
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] microbiomeDataSets_1.1.5       fido_1.0.0
 [3] forcats_0.5.1                  stringr_1.4.0
 [5] dplyr_1.0.8                    purrr_0.3.4
 [7] readr_2.1.2                    tidyr_1.2.0
 [9] tibble_3.1.6                   ggplot2_3.3.5
[11] tidyverse_1.3.1                knitr_1.38
[13] Maaslin2_1.8.0                 ALDEx2_1.26.0
[15] zCompositions_1.4.0-1          truncnorm_1.0-8
[17] NADA_1.6-1.1                   survival_3.3-1
[19] MASS_7.3-56                    ANCOMBC_1.4.0
[21] tidySummarizedExperiment_1.4.1 patchwork_1.1.1
[23] mia_1.3.19                     MultiAssayExperiment_1.20.0
[25] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
[27] XVector_0.34.0                 SingleCellExperiment_1.16.0
[29] SummarizedExperiment_1.24.0    Biobase_2.54.0
[31] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[33] IRanges_2.28.0                 S4Vectors_0.32.4
[35] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[37] matrixStats_0.62.0-9000        BiocStyle_2.22.0
[39] rebook_1.4.0

loaded via a namespace (and not attached):
  [1] rappdirs_0.3.3                coda_0.19-4
  [3] bit64_4.0.5                   irlba_2.3.5
  [5] DelayedArray_0.20.0           data.table_1.14.2
  [7] KEGGREST_1.34.0               RCurl_1.98-1.6
  [9] generics_0.1.2                ScaledMatrix_1.2.0
 [11] microbiome_1.16.0             RSQLite_2.2.12
 [13] bit_4.0.4                     tzdb_0.3.0
 [15] httpuv_1.6.5                  xml2_1.3.3
 [17] lubridate_1.8.0               assertthat_0.2.1
 [19] DirichletMultinomial_1.36.0   viridis_0.6.2
 [21] xfun_0.30                     hms_1.1.1
 [23] ggdist_3.1.1                  promises_1.2.0.1
 [25] evaluate_0.15                 DEoptimR_1.0-11
 [27] fansi_1.0.3                   dbplyr_2.1.1
 [29] readxl_1.4.0                  igraph_1.3.1
 [31] DBI_1.1.2                     htmlwidgets_1.5.4
 [33] tensorA_0.36.2                hash_2.2.6.2
 [35] ellipsis_0.3.2                backports_1.4.1
 [37] bookdown_0.26                 permute_0.9-7
 [39] sparseMatrixStats_1.6.0       vctrs_0.4.1
```

```
 [41] abind_1.4-5                  tidybayes_3.0.2
 [43] cachem_1.0.6                 withr_2.5.0
 [45] robustbase_0.95-0            checkmate_2.0.0
 [47] vegan_2.6-2                  treeio_1.18.1
 [49] getopt_1.20.3                cluster_2.1.3
 [51] ExperimentHub_2.2.1          ape_5.6-2
 [53] dir.expiry_1.2.0             lazyeval_0.2.2
 [55] crayon_1.5.1                 pkgconfig_2.0.3
 [57] labeling_0.4.2               nlme_3.1-157
 [59] vipor_0.4.5                  rlang_1.0.2
 [61] lifecycle_1.0.1              filelock_1.0.2
 [63] BiocFileCache_2.2.1          phyloseq_1.38.0
 [65] modelr_0.1.8                 rsvd_1.0.5
 [67] AnnotationHub_3.2.2          cellranger_1.1.0
 [69] distributional_0.3.0         graph_1.72.0
 [71] Matrix_1.4-1                 lpsymphony_1.22.0
 [73] Rhdf5lib_1.16.0              reprex_2.0.1
 [75] beeswarm_0.4.0               png_0.1-7
 [77] viridisLite_0.4.0            bitops_1.0-7
 [79] rhdf5filters_1.6.0           blob_1.2.3
 [81] DelayedMatrixStats_1.16.0    decontam_1.14.0
 [83] DECIPHER_2.22.0              beachmat_2.10.0
 [85] scales_1.2.0                 memoise_2.0.1
 [87] magrittr_2.0.3               plyr_1.8.7
 [89] zlibbioc_1.40.0              compiler_4.1.3
 [91] RColorBrewer_1.1-3           cli_3.2.0
 [93] ade4_1.7-19                  pbapply_1.5-0
 [95] mgcv_1.8-40                  tidyselect_1.1.2
 [97] stringi_1.7.6                highr_0.9
 [99] yaml_2.3.5                   BiocSingular_1.10.0
[101] svUnit_1.0.6                 ggrepel_0.9.1
[103] grid_4.1.3                   tools_4.1.3
[105] parallel_4.1.3               rstudioapi_0.13
[107] foreach_1.5.2                logging_0.10-108
[109] optparse_1.7.1               gridExtra_2.3
[111] posterior_1.2.1              farver_2.1.0
[113] Rtsne_0.16                   RcppZiggurat_0.1.6
[115] digest_0.6.29                BiocManager_1.30.16
[117] shiny_1.7.1                  Rcpp_1.0.8.3
[119] broom_0.8.0                  scuttle_1.4.0
[121] later_1.3.0                  BiocVersion_3.14.0
[123] AnnotationDbi_1.56.2         httr_1.4.2
[125] Rdpack_2.3                   colorspace_2.0-3
[127] rvest_1.0.2                  XML_3.99-0.9
[129] fs_1.5.2                     splines_4.1.3
[131] yulab.utils_0.0.4            tidytree_0.3.9
```

```
[133] scater_1.22.0                 multtest_2.50.0
[135] plotly_4.10.0                 xtable_1.8-4
[137] jsonlite_1.8.0                nloptr_2.0.0
[139] CodeDepends_0.6.5             Rfast_2.0.6
[141] R6_2.5.1                      mime_0.12
[143] pillar_1.7.0                  htmltools_0.5.2
[145] glue_1.6.2                    fastmap_1.1.0
[147] BiocParallel_1.28.3           BiocNeighbors_1.12.0
[149] interactiveDisplayBase_1.32.0 codetools_0.2-18
[151] pcaPP_1.9-74                  mvtnorm_1.1-3
[153] utf8_1.2.2                    lattice_0.20-45
[155] arrayhelpers_1.1-0            curl_4.3.2
[157] ggbeeswarm_0.6.0              biglm_0.9-2.1
[159] rmarkdown_2.13                biomformat_1.22.0
[161] munsell_0.5.0                 rhdf5_2.38.1
[163] GenomeInfoDbData_1.2.7        iterators_1.0.14
[165] haven_2.5.0                   reshape2_1.4.4
[167] gtable_0.3.0                  rbibutils_2.2.8
```

Chapter 13

Multi-assay analyses

```
## Loading required package: ecodist
```

```
library(mia)
```

Multi-omics means that we integrate data from multiple sources. For example, we can integrate microbial abundances in the gut with biomolecular profiling data from blood samples. This kind of integrative multi-omic approaches can support the analysis of microbiome dysbiosis and facilitate the discovery of novel biomarkers for health and disease.

With cross-correlation analysis, we can analyze how strongly and how differently variables are associated between each other. For instance, we can analyze if higher presence of a specific taxon equals to higher levels of a biomolecule.

The data containers that the *miaverse* utilizes are scalable and they can contain different types of data in a same container. Because of that, the *miaverse* is well-suitable for multi-assay microbiome data which incorporates different types of complementary data sources in a single reproducible workflow.

Another experiment can be stored in altExp slot of SE data container or both experiments can be stored side-by-side in MAE data container.

Different experiments are first imported into SE or TreeSE data container similarly to the case when only one experiment is present. After that different experiments are combined into the same data container. Result is one TreeSE object with alternative experiment in altExp slot, or MAE object with multiple experiment in its experiment slot.

As an example data, we use data from following publication: Hintikka L *et al.* (2021) Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: associating taxonomic and metabolomic patterns in fecal microbiotas with biclustering.

In this article, mice were fed with high-fat and low-fat diets with or without prebiotics. The purpose of this was to study if prebiotics would reduce the negative impacts of high-fat diet.

This example data can be loaded from microbiomeDataSets. The data is already in MAE format. It includes three different experiments: microbial abundance data, metabolite concentrations, and data about different biomarkers. Help for importing data into SE object you can find from here.

```
# Load the data
mae <- microbiomeDataSets::HintikkaXOData()

mae
```

```
## A MultiAssayExperiment object of 3 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 3:
##  [1] microbiota: SummarizedExperiment with 12706 rows and 40 columns
##  [2] metabolites: SummarizedExperiment with 38 rows and 40 columns
##  [3] biomarkers: SummarizedExperiment with 39 rows and 40 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

```
if(!require(stringr)){
    install.packages("stringr")
    library(stringr)
}
# Drop off those bacteria that do not include information in
 ↪  Phylum or lower levels
mae[[1]] <- mae[[1]][!is.na(rowData(mae[[1]])$Phylum), ]
# Clean taxonomy data, so that names do not include additional
 ↪  characters
rowData(mae[[1]]) <- DataFrame(apply(rowData(mae[[1]]), 2,
                                     str_remove, pattern =
 ↪  "._[0-9]__"))
# Microbiome data
mae[[1]]
```

```
## class: SummarizedExperiment
```

```
## dim: 12613 40
## metadata(0):
## assays(1): counts
## rownames(12613): GAYR01026362.62.2014 CVJT01000011.50.2173 ...
##   JRJTB:03787:02429 JRJTB:03787:02478
## rowData names(7): Phylum Class ... Species OTU
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
```

```
# Metabolite data
mae[[2]]
```

```
## class: SummarizedExperiment
## dim: 38 40
## metadata(0):
## assays(1): nmr
## rownames(38): Butyrate Acetate ... Malonate 1,3-dihydroxyacetone
## rowData names(0):
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
```

```
# Biomarker data
mae[[3]]
```

```
## class: SummarizedExperiment
## dim: 39 40
## metadata(0):
## assays(1): signals
## rownames(39): Triglycerides_liver CLSs_epi ... NPY_serum Glycogen_liver
## rowData names(0):
## colnames(40): C1 C2 ... C39 C40
## colData names(0):
```

13.1 Cross-correlation Analysis

Next we can do the cross-correlation analysis. Here we analyse if individual bacteria genera correlates with concentrations of individual metabolites. This helps as to answer the question: "If this bacteria is present, is this metabolite's concentration then low or high"?

```
# Agglomerate microbiome data at Genus level
mae[[1]] <- agglomerateByPrevalence(mae[[1]], rank = "Genus")
# Does log10 transform for microbiome data
mae[[1]] <- transformSamples(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,
                                               abund_values1 =
                                                ↪  "log10",
                                               abund_values2 =
                                                ↪  "nmr",
                                               method =
                                                ↪  "spearman",
                                               p_adj_threshold =
                                                ↪  0.05,
                                               cor_threshold = 0,
                                               # Remove when mia
                                                ↪  is fixed
                                               # mode = "matrix",
                                               # sort = TRUE,
                                               show_warnings =
                                                ↪  FALSE)
```

Creates the heatmap

```
# if( !require("ComplexHeatmap") ){
#     BiocManager::install("ComplexHeatmap")
#     library("ComplexHeatmap")
# }
#
# # Create a heatmap and store it
# plot <- Heatmap(correlations$cor,
#                 # Print values to cells
#                 cell_fun = function(j, i, x, y, width, height,
 ↪  fill) {
#                     # If the p-value is under threshold
#                     if( correlations$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


library(ggplot2)

# Determines the scaling of colors
limits <- c(-1, 1)
breaks <- seq(from = min(limits), to = max(limits), by = 0.2)
colours <- c("darkblue", "blue", "white", "red", "darkred")

# Which observation have p-value under 0.05? --> creates a subset
cor_table_sub <- correlations[which(correlations[["p_adj"]] <
 ↪  0.05), ]

# Creates a ggplot object
ggplot(correlations, aes(x = Var1, y = Var2, fill = cor)) +
  geom_tile() +

  scale_fill_gradientn(name = "Correlation",
                       breaks = breaks, limits = limits, colours
                        ↪  = colours) +

  # Adds label to those observations that have p-value under 0.05
  geom_text(data = cor_table_sub, aes(x = Var1, y = Var2, label =
   ↪  "+")) +

  theme(text = element_text(size=10),
        axis.text.x = element_text(angle=45, hjust=1),
        legend.key.size = unit(1, "cm")) +
  labs(x = "Taxa", y = "Metabolites")
```

13.2 Multi-Omics Factor Analysis

Multi-Omics Factor Analysis (Argelaguet, 2018) (MOFA) is an unsupervised method for integrating multi-omic data sets in a downstream analysis. It could be seen as a generalization of principal component analysis. Yet, with the ability to infer a latent (low-dimensional) representation, shared among the mutliple (-omics) data sets in hand.

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

```
if(!require(MOFA2)){
    BiocManager::install("MOFA2")
}

# For inter-operability between Python and R, and setting Python
 ↪  dependencies,
# reticulate package is needed
if(!require(reticulate)){
    install.packages("reticulate")
```

```
}

reticulate::install_miniconda(force = TRUE)
```

```
## [1] "/github/home/.local/share/r-miniconda"
```

```
reticulate::use_miniconda(condaenv = "env1", required = FALSE)
reticulate::py_install(packages = c("mofapy2"), pip = TRUE)
```

The `mae` object could be used straight to create the MOFA model. Yet, we transform our assays since the model assumes normality per default. Other distributions that can be used, include Poisson or Bernoulli.

```
library(MOFA2)
# For simplicity, classify all high-fat diets as high-fat, and
 ↪  all the low-fat
# diets as low-fat diets
colData(mae)$Diet <- ifelse(colData(mae)$Diet == "High-fat" |
                              colData(mae)$Diet == "High-fat +
                               ↪  XOS",
                            "High-fat", "Low-fat")

# Removing duplicates at the microbiome data
# which are also in form e.g. "Ambiguous" and "uncultured" taxa
mae[[1]] <- mae[[1]][!duplicated(rownames(assay(mae[[1]]))), ]

# Transforming microbiome data with rclr
mae[[1]] <- transformCounts(mae[[1]], abund_values = "counts",
 ↪  method = "rclr")

# Transforming metabolomic data with log10
mae[[2]] <- transformSamples(mae[[2]], abund_values = "nmr",
 ↪  method = "log10")

# Transforming biomarker data with z-transform
mae[[3]] <- transformFeatures(mae[[3]], abund_values = "signals",
 ↪  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
```

```r
# 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): 136 38 39
##  Number of groups: 2
##  Groups names: High-fat Low-fat
##  Number of samples (per group): 20 20
##
```

Model options could be defined as follows:

```r
model_opts <- get_default_model_options(model)
model_opts$num_factors <- 15
head(model_opts)
```

```
## $likelihoods
##  microbiota metabolites  biomarkers
##  "gaussian"  "gaussian"  "gaussian"
##
## $num_factors
## [1] 15
##
## $spikeslab_factors
## [1] FALSE
##
## $spikeslab_weights
## [1] TRUE
##
## $ard_factors
## [1] TRUE
##
## $ard_weights
## [1] TRUE
```

Model's training options are defined with the following:

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

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

Preparing and training the model:

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

Visualizing the variance explained:

```
library(patchwork)
wrap_plots(
    plot_variance_explained(model.trained, x="view", y="factor",
     ↪  plot_total = T),
    nrow = 2
) + plot_annotation(title = "Variance Explained per factor and
 ↪  assay",
                    theme = theme(plot.title = element_text(hjust
                     ↪  = 0.5)))
```

The top weights for each assay using the three first factors:

```
plots <- lapply(c("microbiota", "metabolites","biomarkers"),
 ↪  function(name) {
    plot_top_weights(model.trained,
                     view = name,
                     factors = 1:3,
                     nfeatures = 10) +
        labs(title = paste0("Top weights of the ", name,"
         ↪  assay"))
})
wrap_plots(plots, nrow = 3) & theme(text = element_text(size =
 ↪  8))
```

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

Session Info

View session info

```
R version 4.1.3 (2022-03-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.4 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
```

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

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

other attached packages:
 [1] patchwork_1.1.1                reticulate_1.24
 [3] MOFA2_1.4.0                    ggplot2_3.3.5
 [5] stringr_1.4.0                  microbiomeDataSets_1.1.5
 [7] mia_1.3.19                     MultiAssayExperiment_1.20.0
 [9] TreeSummarizedExperiment_2.1.4 Biostrings_2.62.0
[11] XVector_0.34.0                 SingleCellExperiment_1.16.0
[13] SummarizedExperiment_1.24.0    Biobase_2.54.0
[15] GenomicRanges_1.46.1           GenomeInfoDb_1.30.1
[17] IRanges_2.28.0                 S4Vectors_0.32.4
[19] BiocGenerics_0.40.0            MatrixGenerics_1.6.0
[21] matrixStats_0.62.0-9000        ecodist_2.0.7
[23] BiocStyle_2.22.0               rebook_1.4.0

loaded via a namespace (and not attached):
  [1] AnnotationHub_3.2.2           corrplot_0.92
  [3] BiocFileCache_2.2.1           plyr_1.8.7
  [5] lazyeval_0.2.2                splines_4.1.3
  [7] BiocParallel_1.28.3           scater_1.22.0
  [9] digest_0.6.29                 yulab.utils_0.0.4
 [11] htmltools_0.5.2               viridis_0.6.2
 [13] fansi_1.0.3                   magrittr_2.0.3
 [15] memoise_2.0.1                 ScaledMatrix_1.2.0
 [17] cluster_2.1.3                 DECIPHER_2.22.0
 [19] colorspace_2.0-3              blob_1.2.3
 [21] rappdirs_0.3.3                ggrepel_0.9.1
 [23] xfun_0.30                     dplyr_1.0.8
 [25] crayon_1.5.1                  RCurl_1.98-1.6
 [27] jsonlite_1.8.0                graph_1.72.0
 [29] ape_5.6-2                     glue_1.6.2
 [31] gtable_0.3.0                  zlibbioc_1.40.0
 [33] DelayedArray_0.20.0           BiocSingular_1.10.0
 [35] Rhdf5lib_1.16.0               HDF5Array_1.22.1
 [37] scales_1.2.0                  pheatmap_1.0.12
 [39] DBI_1.1.2                     Rcpp_1.0.8.3
 [41] viridisLite_0.4.0             xtable_1.8-4
```

```
 [43] decontam_1.14.0               tidytree_0.3.9
 [45] bit_4.0.4                     rsvd_1.0.5
 [47] httr_1.4.2                    RColorBrewer_1.1-3
 [49] dir.expiry_1.2.0              ellipsis_0.3.2
 [51] farver_2.1.0                  pkgconfig_2.0.3
 [53] XML_3.99-0.9                  scuttle_1.4.0
 [55] uwot_0.1.11                   CodeDepends_0.6.5
 [57] dbplyr_2.1.1                  here_1.0.1
 [59] utf8_1.2.2                    labeling_0.4.2
 [61] tidyselect_1.1.2              rlang_1.0.2
 [63] reshape2_1.4.4                later_1.3.0
 [65] AnnotationDbi_1.56.2          munsell_0.5.0
 [67] BiocVersion_3.14.0            tools_4.1.3
 [69] cachem_1.0.6                  cli_3.2.0
 [71] DirichletMultinomial_1.36.0   generics_0.1.2
 [73] RSQLite_2.2.12                ExperimentHub_2.2.1
 [75] evaluate_0.15                 fastmap_1.1.0
 [77] yaml_2.3.5                    knitr_1.38
 [79] bit64_4.0.5                   purrr_0.3.4
 [81] KEGGREST_1.34.0               nlme_3.1-157
 [83] sparseMatrixStats_1.6.0       mime_0.12
 [85] compiler_4.1.3                beeswarm_0.4.0
 [87] filelock_1.0.2                curl_4.3.2
 [89] png_0.1-7                     interactiveDisplayBase_1.32.0
 [91] treeio_1.18.1                 tibble_3.1.6
 [93] stringi_1.7.6                 basilisk.utils_1.6.0
 [95] highr_0.9                     forcats_0.5.1
 [97] lattice_0.20-45               Matrix_1.4-1
 [99] vegan_2.6-2                   permute_0.9-7
[101] vctrs_0.4.1                   rhdf5filters_1.6.0
[103] pillar_1.7.0                  lifecycle_1.0.1
[105] BiocManager_1.30.16           BiocNeighbors_1.12.0
[107] cowplot_1.1.1                 bitops_1.0-7
[109] irlba_2.3.5                   httpuv_1.6.5
[111] R6_2.5.1                      bookdown_0.26
[113] promises_1.2.0.1              gridExtra_2.3
[115] vipor_0.4.5                   codetools_0.2-18
[117] MASS_7.3-56                   assertthat_0.2.1
[119] rhdf5_2.38.1                  rprojroot_2.0.3
[121] withr_2.5.0                   GenomeInfoDbData_1.2.7
[123] mgcv_1.8-40                   parallel_4.1.3
[125] grid_4.1.3                    beachmat_2.10.0
[127] basilisk_1.6.0                tidyr_1.2.0
[129] rmarkdown_2.13                DelayedMatrixStats_1.16.0
[131] Rtsne_0.16                    shiny_1.7.1
[133] ggbeeswarm_0.6.0
```

Part III

Appendix

Chapter 14

Resources

14.1 Data containers

14.1.1 Resources for TreeSummarizedExperiment

- SingleCellExperiment (Lun and Risso, 2020)
  - Online tutorial
  - Project page
- SummarizedExperiment (Morgan et al., 2020)
  - Online tutorial
  - Project page
- TreeSummarizedExperiment (Huang, 2020)
  - Online tutorial
  - Project page

14.1.2 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.
- GenomicRanges 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.

14.1.3 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 `MultiAssayExperiment` 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.

14.1.4 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
- microbiome tutorial
- microbiomeutilities
- Bioconductor Workflow for Microbiome Data Analysis: from raw reads to community analyses (Callahan et al. F1000, 2016).

14.2 R programming resources

If you are new to R, you could try swirl for a kickstart to R programming. Further support resources are available through the Bioconductor project.

- R programming basics: Base R
- Basics of R programming: Base R
- R cheat sheets

- R visualization with ggplot2
- R graphics cookbook

Rmarkdown

- Rmarkdown tips

RStudio

- RStudio cheat sheet

14.2.1 Bioconductor Classes

S4 system

S4 class system has brought several useful features to the object-oriented programming paradigm within R, and it is constantly deployed in R/Bioconductor packages.

Online Document:

- Hervé Pagès, A quick overview of the S4 class system.
- Laurent Gatto, A practical tutorial on S4 programming
- John M. Chambers. How S4 Methods Work

Books:

- John M. Chambers. Software for Data Analysis: Programming with R. Springer, New York, 2008. ISBN-13 978-0387759357.
- I Robert Gentleman. R Programming for Bioinformatics. Chapman & Hall/CRC, New York, 2008. ISBN-13 978-1420063677

Figure sources:

Original article - Huang R *et al.* (2021) TreeSummarizedExperiment: a S4 class for data with hierarchical structure. F1000Research 9:1246.

Reference Sequence slot extension - Lahti L *et al.* (2020) Upgrading the R/Bioconductor ecosystem for microbiome research F1000Research 9:1464 (slides).

Chapter 15

Extra material

15.1 Interactive 3D Plots

```
# Installing required packages
if (!require(rgl)){
  BiocManager::install("rgl")
}
if (!require(plotly)){
  BiocManager::install("plotly")
}
```

```
library(knitr)
library(rgl)
knitr::knit_hooks$set(webgl = hook_webgl)
```

In this section we make a 3D version of the earlier Visualizing the most dominant genus on PCoA, with the help of the plotly R package.

```
# Installing the package
if (!require(curatedMetagenomicData)){
  BiocManager::install("curatedMetagenomicData")
}
# Importing necessary libraries
library(curatedMetagenomicData)
library(dplyr)
library(DT)
library(mia)
```

```
library(scater)

# Querying the data
tse <- sampleMetadata %>%
    filter(age >= 18) %>% # taking only data of age 18 or above
    filter(!is.na(alcohol)) %>% # excluding missing values
    returnSamples("relative_abundance")

tse_Genus <- agglomerateByRank(tse, rank="genus")
tse_Genus <-
 ↪  addPerSampleDominantTaxa(tse_Genus,abund_values="relative_abundance",
 ↪  name = "dominant_taxa")

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

# Getting the 6 top taxa
top_taxa <- getTopTaxa(tse_Genus,top = 6, abund_values =
 ↪  "relative_abundance")

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

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

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

# Retrieving the explained variance
e <- attr(reducedDim(tse_Genus, "PCoA_BC"), "eig");
var_explained <- e/sum(e[e>0])*100
```

Interactive 3D visualization of the most dominant genus on PCoA. Note that labels at legend can be used to visualize one or more Genus separately (double click to isolate one from the others, or toggle to select multiple ones).

```
library(plotly)

# 3D Visualization
reduced_data  <- as.data.frame(reducedDim(tse_Genus)[,])
names(reduced_data) <- c("PC1","PC2","PC3")
plot_ly(reduced_data, x=~PC1,y=~PC2,z=~PC3)%>%

   ↪  add_markers(color=sapply(strsplit(colData(tse_Genus)$most_abundant,
   ↪  "_"), tail, 1), size=5,
              colors=c("black", "blue", "lightblue", "darkgray",
               ↪  "magenta", "darkgreen", "red")) %>%
  layout(scene=list(xaxis=list(title = paste("PC1
   ↪  (",round(var_explained[1],1),"%)")),
                    yaxis=list(title = paste("PC2
                     ↪  (",round(var_explained[2],1),"%)")),
                    zaxis=list(title = paste("PC3
                     ↪  (",round(var_explained[3],1),"%)"))))
```

WebGL is not supported by your browser - visit https://get.webgl.org for more info

Chapter 16

Acknowledgments

This work would not have been possible without the countless contributions and interactions within the broader research community. 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 application fields (Gentleman et al., 2004, Huber et al. (2015)).

The base ecosystem of data containers, packages, and tutorials was set up as a collaborative effort by Tuomas Borman, Henrik Eckermann, Chouaib Benchraka, Chandler Ross, Shigdel Rajesh, Yağmur Şimşek, Giulio Benedetti, Sudarshan Shetty, Felix Ernst, and Leo Lahti, with further support from the COST Action network on Statistical and Machine Learning Techniques for Human Microbiome Studies (ML4microbiome) (Moreno-Indias et al., 2021). The framework is based on the *TreeSummarizedExperiment* data container created by Ruizhu Huang and others (Huang, 2020). The idea of using this container as a new 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).

Ample demonstration data resources have been made available as the curatedMetagenomicData project by Edoardo Pasolli, Lucas Schiffer, Levi Waldron and others (Pasolli et al., 2017) adding important early support for the emerging framework. A number of other contributors have advanced the ecosystem further, and will be acknowledged in the individual packages, pull requests, issues, and other work.

The work has drawn inspiration from many sources, most notably from the work on *phyloseq* by Paul McMurdie and Susan Holmes (McMurdie and Holmes, 2013) who pioneered the work on rigorous and reproducible microbiome data science ecosystems in R/Bioconductor. The phyloseq framework continues to provide

a strong complementary set of packages and methods for microbiome studies, and we do our best to support full interoperability.

The open source books by Susan Holmes and Wolfgang Huber, Modern Statistics for Modern Biology (Holmes and Huber, 2019) and by Garret Grolemund and Hadley Wickham, the R for Data Science (Grolemund and Wickham, 2017), and Richard McElreath's Statistical Rethinking and the associated online resources by Solomon Kurz (McElreath, 2020) are key references that advanced reproducible data science training and dissemination. The Orchestrating Single-Cell Analysis with Bioconductor, or *OSCA* book by Robert Amezquita, Aaron Lun, Stephanie Hicks, and Raphael Gottardo (Amezquita et al., 2020a) has implemented closely related work on the *SummarizedExperiment* data container and its derivatives in the field of single cell sequencing studies. Many approaches used in this book have been derived from the OSCA framework, with various adjustments and extensions dedicated to microbiome data science.

Bibliography

A, C. (1984). Non-parametric estimation of the number of classes in a population. *Scandinavian Journal of Statistics*, 11(4):265–270.

A, C. and SM, L. (1992). Estimating the number of classes via sample coverage. *Journal of the American statistical Association*, 87(417):210–217.

Amezquita, R., Lun, A., Hicks, S., and Gottardo, R. (2020a). *Orchestrating Single-Cell Analysis with Bioconductor*. Bioconductor.

Amezquita, R. A., Lun, A. T., Becht, E., Carey, V. J., Carpp, L. N., Geistlinger, L., Marini, F., Rue-Albrecht, K., Risso, D., Soneson, C., Waldron, L., Pagès, H., Smith, M. L., Huber, W., Morgan, M., Gottardo, R., and Hicks, S. C. (2020b). Orchestrating single-cell analysis with bioconductor. *Nature Methods*, 17:137–145.

Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, 26(1):32–46.

Anderson, M. J. (2006). Distance-based tests for homogeneity of multivariate dispersions. *Biometrics*, 62:245–253.

Argelaguet, R. e. a. (2018). Multi-omics factor analysis—a framework for unsupervised integration of multi-omics data sets. *Molecular Systems Biology*, 14(6):e8124.

Callahan, B., McMurdie, P., and Holmes, S. (2020). *dada2: Accurate, high-resolution sample inference from amplicon sequencing data*. R package version 1.18.0.

Callahan, B. J., Sankaran, K., Fukuyama, J. A., McMurdie, P. J., and Holmes, S. P. (2016). Bioconductor workflow for microbiome data analysis: from raw reads to community analyses [version 2; peer review: 3 approved]. *F1000Research*, 5:1492.

Davis, N. M., Proctor, D. M., Holmes, S. P., Relman, D. A., and Callahan, B. J. (2018). Simple statistical identification and removal of contaminant sequences in marker-gene and metagenomics data. *Microbiome*, 6(1):1–14.

Ernst, F. G., Shetty, S., and Lahti, L. (2020). *mia: Microbiome analysis*. R package version 0.98.15.

Faith, D. P. (1992). Conservation evaluation and phylogenetic diversity. *Biological Conservation*, 61(1):10.

Gentleman, R. 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.

Grolemund, G. and Wickham, H. (2017). *R for Data Science*. O'Reilly.

Holmes, S. and Huber, W. (2019). *Modern Statistics for Modern Biology*. Cambridge University Press, New York, NY.

Huang, R. (2020). *TreeSummarizedExperiment: a S4 Class for Data with Tree Structures*. R package version 1.6.2.

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.

Lahti, L., Salojärvi, J., Salonen, A., Scheffer, M., and de Vos, W. M. (2014). Tipping elements in the human intestinal ecosystem. *Nature Communications*, 2014:1–10.

Lahti, L., Shetty, S., Ernst, F. M., et al. (2021). *Orchestrating Microbiome Analysis with Bioconductor [beta version]*.

Lin, H. and Peddada, S. D. (2020). Analysis of compositions of microbiomes with bias correction. *Nat Commun*, 11(1):3514.

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.

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).

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. J. 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.

Nearing, J. T., Douglas, G. M., Hayes, M., MacDonald, J., Desai, D., Allward, N., Jones, C. M. A., Wright, R., Dhanani, A., Comeau, A. M., and Langille, M. G. I. (2021). Microbiome differential abundance methods produce disturbingly different results across 38 datasets. Preprint, Bioinformatics.

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.

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.

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. and Waldron, L. (2020). *MultiAssayExperiment: Software for the integration of multi-omics experiments in Bioconductor*. R package version 1.16.0.

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_res Shetty-MDS.pdf.

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.