Introduction to microbiome data science

Leo Lahti, Tuomas Borman, Henrik Eckermann, and Chouaib Benchraka

2021-07-08

Contents

Chapter 1

Overview

Welcome to Radboud Summer School, July 2021

Figure source: Moreno-Indias *et al.* (2021) Statistical and Machine Learning Techniques in Human Microbiome Studies: Contemporary Challenges and Solutions. Frontiers in Microbiology 12:11.

1.1 The miaverse framework

The *miaverse* (mia = **MI**crobiome **A**nalysis) is an R/Bioconductor framework for microbiome data science. It aims to extend the capabilities of another popular framework, phyloseq.

The miaverse framework consists of an efficient data structure, an associated package ecosystem, demonstration data sets, and open documentation. These are explained in more detail in the online book Orchestrating Microbiome Analysis.

This training material walks you through an example workflow that shows the standard steps of taxonomic data analysis covering data access, exploration, analysis, visualization and reporoducible reporting. **You can run the workflow by simply copy-pasting the examples.** For advanced material, you can test and modify further examples from the OMA book, or try to apply the techniques to your own data.

1.2 Learning goals

This course provides an overview of the standard bioinformatics workflow in taxonomic profiling studies, ranging from data preprocessing to statistical analysis and reproducible reporting, with a focus on examples from human gut microbiota studies. You will become familiar with standard bioinformatics concepts and methods in taxonomic profiling studies of the human microbiome. This includes better understanding of the specific statistical challenges, practical hands-on experience with the commonly used methods, and reproducible research with R.

After the course you will know how to approach new tasks in microbiome data science by utilizing available documentation and R tools.

Target audience Advanced students and applied researchers who wish to develop their skills in microbial community analysis.

Venue Radboud University / Online, Nijmegen. July 5-16, 2021, with contributions by University of Turku, Finland.

1.3 Acknowledgments

Citation "Introduction to miaverse (2021). Tuomas Borman, Felix Ernst, Sudarshan Shetty, Henrik Eckermann, Leo Lahti. URL: https://microbiome.github.io."

Contact

- Leo Lahti, University of Turku
- miaverse collective

Source code

The source code of this repository is fully reproducible and contains the Rmd files with executable code. All files can be rendered at one go by running the file main.R. You can check the file for details on how to clone the repository and convert it into a gitbook, although this is not necessary for the training.

- Source code (github): miaverse teaching material
- Course page (html): miaverse teaching material

Chapter 2

Program

The course takes place on each working day from 9am – 1pm (CEST). Short breaks will be scheduled between sessions.

2.1 Monday 12 July: from raw sequences to ecological data analysis

Lectures

- Microbiota analysis: association studies vs. causality; microbiota sequencing methods (16S, shotgun, metagenomics) - by dr. Tom Ederveen (Radboud UMC Nijmegen, The Netherlands)
- DNA isolation and 16S rRNA gene sequencing; bioinformatics step 1: from raw sequences to OTU table in a biom file – by Tom Ederveen (Radboudumc Nijmegen, The Netherlands)

Demonstration & Practical

- Importing the data to R environment for interactive data analysis – by prof. dr. Leo Lahti (University of Turku, Finland)

2.2 Tuesday 13 July - Alpha diversity

Lecture

- Key concepts in microbiome data science

Practical

- Alpha diversity: estimation, analysis, and visualization

2.3 Wednesday 14 July - Beta diversity

Demonstration

- Community similarity

Practical

- Beta diversity: estimation, analysis, and visualization

2.4 Thursday 15 July - Differential abundance

Lecture

- Differential abundance analysis methods

Practical

- Differential abundance in practice

Lecture

- Overview of microbiota data science methods & concepts

2.5 Friday 16 July: Presentations & closing

Student presentations on microbiome data analytics

Chapter 3

Getting started

3.1 Checklist (before the course)

Install the following software in advance in order to avoid unnecessary delays and leaving more time for the course contents.

- R (version >4.1.0)
- RStudio; choose "Rstudio Desktop" to download the latest version. Optional but preferred. For further details, check the Rstudio home page.
- Install and load the required R packages.
- After a successful installation you can start with the case study examples in this training material

3.2 Support and resources

For additional reading and online material, see Material section

For online support on installation and other matters, you can join us at:

- Users: miaverse Gitter channel
- Developers: Bioconductor Slack #microbiomeexperiment channel (ask for an invitation)

3.3 Installing and loading the required R packages

This section shows how to install and load all required packages into the R session. Only uninstalled packages are installed.

```
# List of packages that we need from cran and bioc
cran_pkg <- c("BiocManager", "bookdown", "dplyr", "ecodist", "ggplot2", "gridExtra", "
bioc_pkg <- c("ANCOMBC", "ape", "DESeq2",  "DirichletMultinomial", "mia", "miaViz")

# Gets those packages that are already installed
cran_pkg_already_installed <- cran_pkg[ cran_pkg %in% installed.packages() ]
bioc_pkg_already_installed <- bioc_pkg[ bioc_pkg %in% installed.packages() ]

# Gets those packages that need to be installed
cran_pkg_to_be_installed <- setdiff(cran_pkg, cran_pkg_already_installed)
bioc_pkg_to_be_installed <- setdiff(bioc_pkg, bioc_pkg_already_installed)
```

```
# If there are packages that need to be installed, installs them from CRAN
if( length(cran_pkg_to_be_installed) ) {
   install.packages(cran_pkg_to_be_installed)
}
```

```
# If there are packages that need to be installed, installs them from Bioconductor
if( length(bioc_pkg_to_be_installed) ) {
   BiocManager::install(bioc_pkg_to_be_installed, ask = F)
}
```

Now all required packages are installed, so let's load them into the session. Some function names occur in multiple packages. That is why miaverse's packages mia and miaViz are prioritized. Packages that are loaded first have higher priority.

```
# Reorders bioc packages, so that mia and miaViz are first
bioc_pkg <- c(bioc_pkg[ bioc_pkg %in% c("mia", "miaViz") ], bioc_pkg[ !bioc_pkg %in% c

# Loading all packages into session. Returns true if package was successfully loaded.
sapply(c(bioc_pkg, cran_pkg), require, character.only = TRUE)
```

```
##                  mia               miaViz              ANCOMBC                  ape
##                 TRUE                 TRUE                 TRUE                 TRUE
##          BiocManager             bookdown                dplyr              ecodist
##                 TRUE                 TRUE                 TRUE                 TRUE
##                knitr                vegan
##                 TRUE                 TRUE
```

Chapter 4

Importing microbiome data

This section demonstrates how to import microbiome profiling data in R.

The *biom* format is a standard file format for microbiome data. Here, we import *biom* data files into a specific data container (structure) in R. Specifically, we use a *TreeSummarizedExperiment* (TSE) data container. This provides the basis for the *miaverse* data science framework.

We use example data from the following publication: Tengeler AC *et al.* (2020) **Gut microbiota from persons with attention-deficit/hyperactivity disorder affects the brain in mice**. Microbiome 8:44. In this study, mice are colonized with microbiota from participants with ADHD (attention deficit hyperactivity disorder) and healthy participants. The aim of the study was to assess whether the mice display ADHD behaviors after being inoculated with ADHD microbiota, suggesting a role of the microbiome in ADHD pathology.

4.1 Data access

Downloading the data You can download the data from data subfolder.

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.

```
se <- loadFromBiom(biom_file_path)
```

4.2 Investigate the R data object

We have now imported the data set in R. Let us investigate its contents.

```
print(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
```

4.2.1 rowData (taxonomic information)

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.

```
knitr::kable(head(rowData(se)))
```

	taxonomy1	taxonomy2	taxonomy3	taxonomy4
1726470	"k__Bacteria	p__Bacteroidetes	c__Bacteroidia	o__Bacteroidales
1726471	"k__Bacteria	p__Bacteroidetes	c__Bacteroidia	o__Bacteroidales
17264731	"k__Bacteria	p__Bacteroidetes	c__Bacteroidia	o__Bacteroidales
17264726	"k__Bacteria	p__Bacteroidetes	c__Bacteroidia	o__Bacteroidales
1726472	"k__Bacteria	p__Verrucomicrobia	c__Verrucomicrobiae	o__Verrucomicrobiales
17264724	"k__Bacteria	p__Bacteroidetes	c__Bacteroidia	o__Bacteroidales

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.

```r
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
knitr::kable(head(rowData(se)))
```

	Kingdom	Phylum	Class	Order	Family	Genu
1726470	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
1726471	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
17264731	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Porphyromonadaceae	Paral
17264726	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
1726472	Bacteria	Verrucomicrobia	Verrucomicrobiae	Verrucomicrobiales	Verrucomicrobiaceae	Akke
17264724	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte

4.2.2 colData (sample information)

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

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

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

Let us add a sample meta data file.

```r
# 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", "sa

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

Now `colData` includes the sample metadata. Use kable to print it more nicely.

```
knitr::kable(head(colData(se)))
```

	patient_status	cohort	patient_status_vs_cohort	sample_name
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

4.2.3 Phylogenetic tree information

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 also includes a `rowTree` slot.

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

# tse includes same data as se
print(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).

```
tree <- ape::read.tree(tree_file_path)

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

# Check
tse
```

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

Now `rowTree` includes a phylogenetic tree:

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

4.3 Further examples

See the online book for more examples on data exploration and manipulation.

Chapter 5

Microbiome data exploration

Now we have loaded the data set into R and confirmed that it has all the necessary components. Next, let us walk through some basic operations for data exploration.

5.1 Investigate

Dimensionality tells us, how many taxa and samples the data contains. As we can see, there are 151 taxa and 27 samples.

```
dim(tse)
```

```
## [1] 151  27
```

The `rowData` slot contains a taxonomic table. This includes taxonomic information for each of the 151 entries. With the `head()` command, we can print just the beginning of the table.

The `rowData` seems to contain information from 6 different taxonomy classes.

```
knitr::kable(head(rowData(tse)))
```

	Kingdom	Phylum	Class	Order	Family	Genu
1726470	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
1726471	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
17264731	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Porphyromonadaceae	Parab
17264726	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte
1726472	Bacteria	Verrucomicrobia	Verrucomicrobiae	Verrucomicrobiales	Verrucomicrobiaceae	Akke
17264724	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacte

The colData slot contains sample metadata. It contains information for all 27 samples. However, here only the 6 first samples are shown as we use the `head()` command. There are 4 columns, that contain information, e.g., about patients' status, and cohort.

```
knitr::kable(head(colData(tse)))
```

	patient_status	cohort	patient_status_vs_cohort	sample_name
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

From here, we can draw summaries of the sample (column) data, for instance to see what is the patient status distribution.

The command `colData(tse)$patient_status` fetches the data from the column, and `table()` creates a table that shows how many times each class is present, and `sort()` sorts the table to ascending order.

There are 13 samples from patients having ADHD, and 14 control samples.

```
sort(table(colData(tse)$patient_status))
```

```
##
##    ADHD Control
##      13      14
```

5.2 Manipulate

5.2.1 Transformations

Microbial abundances are typically 'compositional' (relative) in the current microbiome profiling data sets. This is due to technical aspects of the data generation process (see e.g. Gloor et al., 2017).

The next example calculates relative abundances as these are usually easier to interpret than plain counts. For some statistical models we need to transform the data into other formats as explained in above link (and as we will see later).

```
# Calculates relative abundances, and stores the table to assays
tse <- transformCounts(tse, method = "relabundance")
```

A variety of standard transformations for microbiome data are available for `TSE` data objects through mia R package.

5.2.2 Aggregation

Microbial species can be called at multiple taxonomic resolutions. We can easily agglomerate the data based on taxonomic ranks. Here, we agglomerate the data at Phylum level.

```r
tse_phylum <- agglomerateByRank(tse, rank = "Phylum")

# Show dimensionality
dim(tse_phylum)
```

```
## [1]  5 27
```

Now there are 5 taxa and 27 samples, meaning that there are 5 different Phylum level taxonomic groups. Looking at the `rowData` after agglomeration shows all Firmicutes are combined together, and all lower rank information is lost.

From the assay we can see that all abundances of taxa that belong to Firmicutes are summed up.

```r
knitr::kable(head(rowData(tse_phylum)))
```

	Kingdom	Phylum	Class	Order	Family	Genus
Bacteroidetes	Bacteria	Bacteroidetes	NA	NA	NA	NA
Verrucomicrobia	Bacteria	Verrucomicrobia	NA	NA	NA	NA
Proteobacteria	Bacteria	Proteobacteria	NA	NA	NA	NA
Firmicutes	Bacteria	Firmicutes	NA	NA	NA	NA
Cyanobacteria	Bacteria	Cyanobacteria	NA	NA	NA	NA

If you are sharp, you have by now noticed that all the aggregated values in the above example are NA's (missing data). This is because the agglomeration is missing abundances for certain taxa, and in that case the sum is not defined by default (`na.rm = FALSE`). We can ignore the missing values in summing up the data by setting `na.rm = TRUE`; then the taxa that do not have information in specified level will be removed. Those taxa that do not have information in specified level are agglomerated at lowest possible level that is left after agglomeration.

```r
temp <- rowData(agglomerateByRank(tse, rank = "Genus"))

# Prints those taxa that do not have information at the Genus level
knitr::kable(head(temp[temp$Genus == "",]))
```

	Kingdom	Phylum	Class	Order	Family
Family:Lachnospiraceae	Bacteria	Firmicutes	Clostridia	Clostridiales	Lachnos
Order:Bacteroidales	Bacteria	Bacteroidetes	Bacteroidia	Bacteroidales	
Order:Clostridiales	Bacteria	Firmicutes	Clostridia	Clostridiales	
Family:Enterobacteriaceae	Bacteria	Proteobacteria	Gammaproteobacteria	Enterobacteriales	Enteroba
Order:Gastranaerophilales	Bacteria	Cyanobacteria	Melainabacteria	Gastranaerophilales	

Here agglomeration is done similarly, but na.rm = TRUE

```
temp2 <- rowData(agglomerateByRank(tse, rank = "Genus", na.rm = TRUE))

print(paste0("Agglomeration with na.rm = FALSE: ", dim(temp)[1], " taxa."))
```

```
## [1] "Agglomeration with na.rm = FALSE: 54 taxa."
```

```
print(paste0("Agglomeration with na.rm = TRUE: ", dim(temp2)[1], " taxa."))
```

```
## [1] "Agglomeration with na.rm = TRUE: 49 taxa."
```

The mia package contains further examples on various data agglomeration and splitting options.

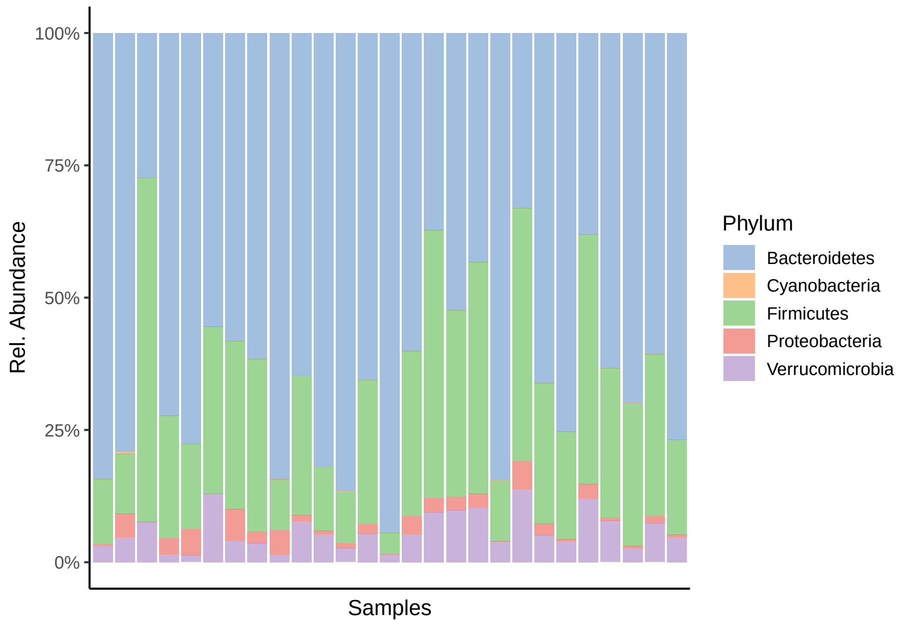
5.3 Visualize

The miaViz package facilitates data visualization. Let us plot the Phylum level abundances.

```
# Here we specify "relabundance" to be abundance table that we use for plotting.
# Note that we can use agglomerated or non-agglomerated tse as an input, because
# the function agglomeration is built-in option.

# Legend does not fit into picture, so its height is reduced.
plot_abundance <- plotAbundance(tse, abund_values="relabundance", rank = "Phylum") +
  theme(legend.key.height = unit(0.5, "cm")) +
  scale_y_continuous(label = scales::percent)
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will replace
```

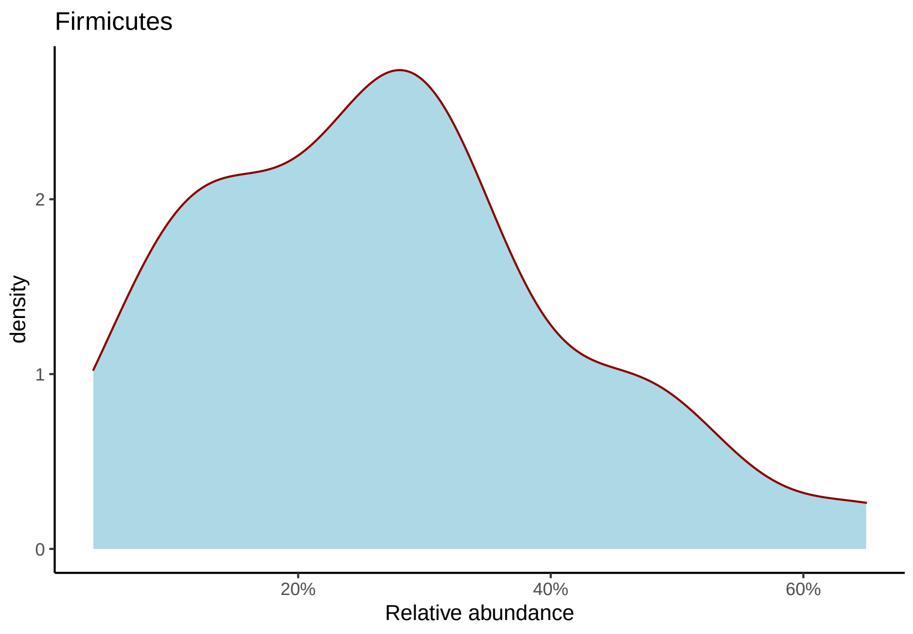
```
plot_abundance
```



Density plot shows the overall abundance distribution for a given taxonomic group. Let us check the relative abundance of Firmicutes across the sample collection. The density plot is a smoothened version of a standard histogram.

The plot shows peak abundances around 30 %.

```r
# Subset data by taking only Firmicutes
tse_firmicutes <- tse_phylum["Firmicutes"]

# Gets the abundance table
abundance_firmicutes <- assay(tse_firmicutes, "relabundance")

# Creates a data frame object, where first column includes abundances
firmicutes_abund_df <- as.data.frame(t(abundance_firmicutes))
# Rename the first and only column
colnames(firmicutes_abund_df) <- "abund"

# Creates a plot. Parameters inside feom_density are optional. With
# geom_density(bw=1000), it is possible to adjust bandwidth.
firmicutes_abund_plot <- ggplot(firmicutes_abund_df, aes(x = abund)) +
  geom_density(color="darkred", fill="lightblue") +
  labs(x = "Relative abundance", title = "Firmicutes") +
  theme_classic() + # Changes the background
  scale_x_continuous(label = scales::percent)

firmicutes_abund_plot
```



```
# # Does the same thing but differently
# # Calculates the density. Bandwidth can be adjusted; here, it is 0.065.
# # density() is from stats package
# density_firmicutes <- density(abundance_firmicutes, bw = 0.065)
#
# # Plots the density
# plot(density_firmicutes,
#      xlab="Relative abundance",
#      ylab="Density",
#      main=paste0("Firmicutes (",density_firmicutes$n, " obs, ", density_firmicutes$b
```

For more visualization options and examples, see the miaViz vignette.

Chapter 6

Alpha diversity

This section demonstrates how alpha diversity indices are calculated.

Alpha diversity is a quantity that measures diversity of taxa in within the sample. Higher numbers of unique taxa, and more even abundance distributions within a sample yield larger values for alpha diversity.

Alpha diversity is an important quantity in a microbiome research. The mia package provides access to a wide variety of alpha diversity indices. As an example, we show how to calculate Shannon and Faith diversity indices.

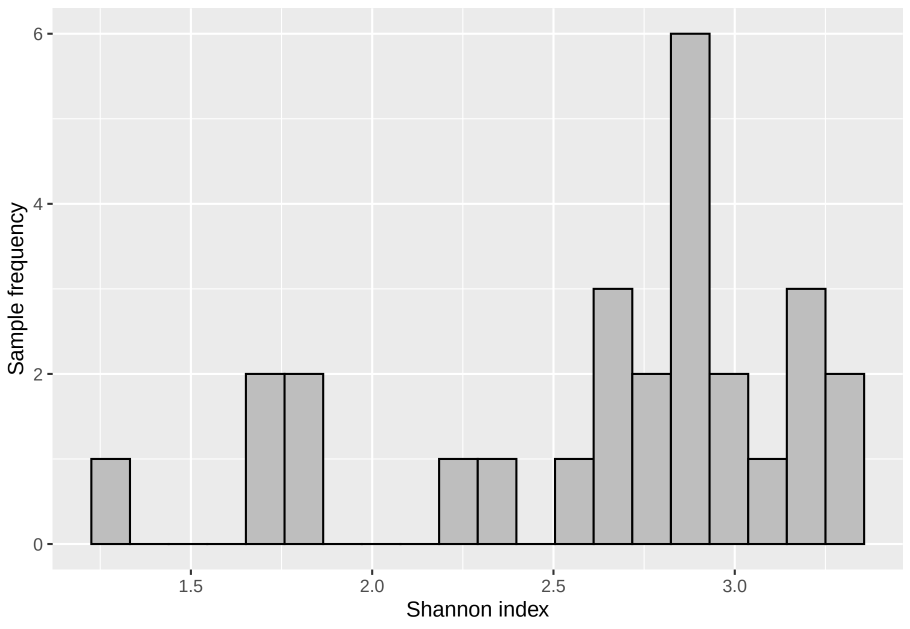
Shannon index reflects how many different taxa there are and how evenly they are distributed within a sample. Faith index additionally takes into account the phylogenetic relations between the taxa when quantifying the overall community diversity. In both indices, higher values represent higher diversity.

```r
# Indices to be calculated. If we don't specify indices, by default, every index
# is calculated.
indices <- c("shannon", "faith")

# Indices are stored in colData (i.e., sample metadata). We can specify the name
# of column, or we can use the default name which is the name of index
# (i.e., "shannon" and "faith").
names <- c("Shannon_index", "Faith_diversity_index")

# Calculates indices
tse <- estimateDiversity(tse, index = indices, name = names)

# Shows the calculated indices
knitr::kable(head(colData(tse)[names]))
```

	Shannon_index	Faith_diversity_index
A110	1.765407	7.39224
A12	2.716438	6.29378
A15	3.178103	6.60608
A19	2.891987	6.79708
A21	2.841979	6.65110
A23	2.797942	5.96246

Next we can visualize Shannon index with histogram.

```
# ggplot needs data.frame as input. Because colData is DataFrame, it needs to be
# converted.
shannon_hist <- ggplot(as.data.frame(colData(tse)),
                       aes(x = Shannon_index)) +
  geom_histogram(bins = 20, fill = "gray", color = "black") +
  labs(x = "Shannon index", y = "Sample frequency")

shannon_hist
```



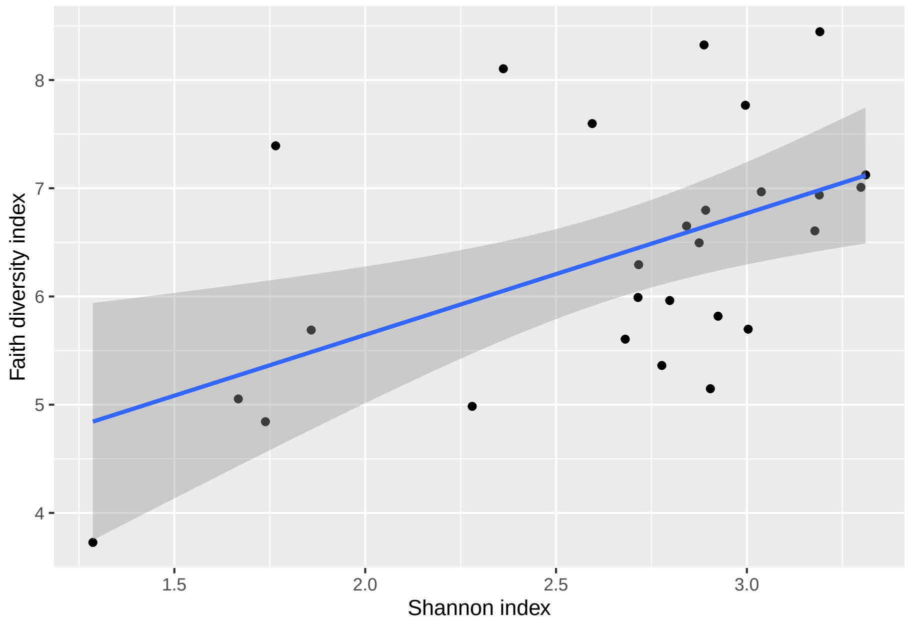
```
# # Same thing but done differently
# # Creates histogram. With "break", number of bins can be specified. However, the
# # value is taken as a suggestion, because hist() uses pretty() to calculate breakpoi
# hist(colData(tse)$Shannon_index, col = "green", breaks = 20,
# xlab = "Shannon index",
# ylab = "Sample frequency",
# main = "Histogram of Shannon index")
```

To see, if there is dependency between Shannon and Faith, we can do cross-plot i.e., scatter plot, where one index is on the x-axis and another on the y-axis.

It seems that, there is a positive correlation between these two indices.

```r
# # Does the same thing but differently
# plot(colData(tse)$Shannon_index, colData(tse)$Faith_diversity_index,
#      xlab = "Shannon index",
#      ylab = "Faith diversity index",
#      main = "plot()") +
#   # Adds regression line
#   abline(lm(colData(tse)$Faith_diversity_index ~ colData(tse)$Shannon_index))

cross_plot <- ggplot2::ggplot(as.data.frame(colData(tse)),
                                     aes(x = Shannon_index, y = Faith_diversity_index)) +
  geom_point() + # Adds points
  geom_smooth(method=lm) + # Adds regression line
  xlab("Shannon index") + # x axis title
  ylab("Faith diversity index")  # y axis title

cross_plot
```

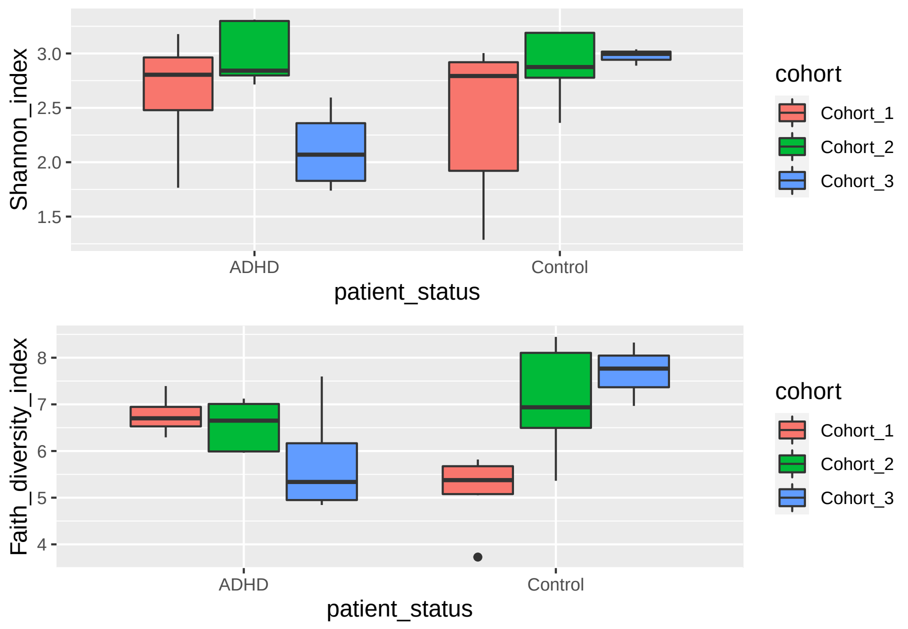
```
## `geom_smooth()` using formula 'y ~ x'
```



6.1 Visualization

Next let's compare indices between different patient status and cohorts. Boxplot is suitable for that purpose.

```
# Creates Shannon boxplot
shannon_box <- ggplot(as.data.frame(colData(tse)), aes(x = patient_status,
                                                        y = Shannon_index, fill = cohor
  geom_boxplot() +
  theme(title = element_text(size = 12)) # makes titles smaller

# Creates Faith boxplot
faith_box <- ggplot(as.data.frame(colData(tse)), aes(x = patient_status,
                                                      y = Faith_diversity_index,
                                                      fill = cohort)) +
  geom_boxplot() +
  theme(title = element_text(size = 12)) # makes titles smaller

# Puts them into same picture
gridExtra::grid.arrange(shannon_box, faith_box, nrow = 2)
```



6.2 Statistical testing and comparisons

To further investigate if patient status explains the variation of Shannon index, let's do a Wilcoxon test.

The Wilcoxon test is similar to a Student's t-test, however, the Student's t-test is parametric. It means that the data must be normally distributed. The Wilcoxon test is non-parametric so it doesn't make any assumptions about the distribution.

The Wilcoxon test tests if there are statistical differences between two groups. Here it tests, if ADHD and control groups have different Shannon index values. As we can see, there is no difference between groups, because the p-value is over 0.05, which is often used as a standard cutoff point.

```r
# Wilcoxon test, where Shannon index is the variable that we are comparing.
# Patient status - ADHD or control - is the factor that we use for grouping.
wilcoxon_shannon <- wilcox.test(Shannon_index ~ patient_status, data = colData(tse))

wilcoxon_shannon
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  Shannon_index by patient_status
## W = 76, p-value = 0.4879
## alternative hypothesis: true location shift is not equal to 0
```

Another test that we can make is to test if ADHD samples differs between different cohorts. From boxplot that we made in previous step, we can see that there might be statistically significant difference between different cohorts.

Let's compare Shannon index of ADHD samples between cohort 2 and cohort 3.

As we can see, there is statistically significant difference between the cohorts.

```r
# Takes subset of colData. Takes only ADHD samples
ADHD_shannon <- colData(tse)[ colData(tse)[, "patient_status"] == "ADHD" , ]

# Takes subset of colData. Takes only samples that are in cohort 2 or cohort 3.
ADHD_shannon <- ADHD_shannon[ ADHD_shannon[, "cohort"] %in% c("Cohort_2", "Cohort_3") , ]

# Wilcoxon test, where Shannon index is the variable that we are comparing.
# Cohort - 2 or 3 - is the factor that we use for grouping.
wilcoxon_shannon_ADHD_cohorts <- wilcox.test(Shannon_index ~ cohort, data = ADHD_shannon)

wilcoxon_shannon_ADHD_cohorts
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  Shannon_index by cohort
## W = 20, p-value = 0.01587
## alternative hypothesis: true location shift is not equal to 0
```

6.3 Further resources

For more examples, see a dedicated section on alpha diversity in the online book.

Chapter 7

Beta diversity

In this section, we go through how to calculate and visualize beta diversity.

Beta diversity reflects the difference in microbial composition between samples. Unlike alpha diversity, where we calculate diversity within a sample, any particular beta diversity value can only be calculated between two samples. There are different ways to measure the distance between two samples. Some of the common choices include Bray-Curtis, Unifrac, Jaccard index, and the Aitchison distance metric. Each of these (dis)similarity measures emphasizes different aspects of (dis)similarity. For example, UniFrac is a distance metric that incorporates phylogenetic information between samples. Or, the Jaccard-index measures the distance based on presence/absence of any taxon irrespective of its abundance.

7.1 Examples of PCoA with different settings

After calculating beta diversity, we end up with a (dis)similarity matrix that contains for each sample the distance to any other sample. To explore our data, we would like to visualize the distances between samples and potentially map meta data into the same graph. This would enable us to explore whether distance between samples could be related to known variables. Given the many dimensions in the matrix, we first need to apply a technique that summarises most of the information into fewer dimensions (components). Principal Coordinate Analysis (PCoA) is a technique that can achieve exactly that.

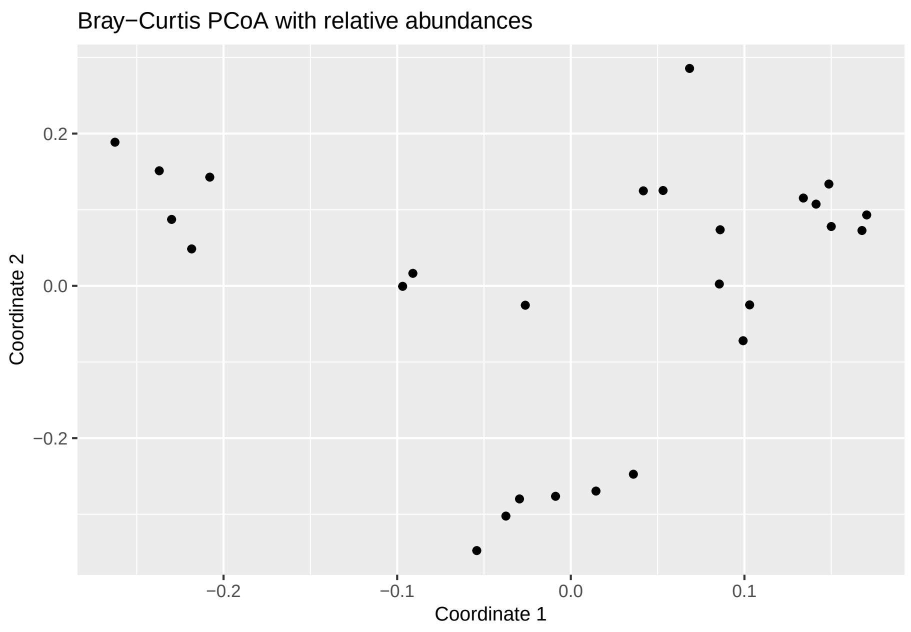
As a side note for those of you familiar with Principal Component Analysis (PCA): A PCoA where you enter a matrix of Euclidean distances will be similar to a PCA of the microbial abundances (except for potential scaling differences).

Back to our PCoA: Typically, we retain 2-3 components that contain most of the information in the data. Note however, that the other components might contain important information as well. After all these steps, we thus end up with 2-3 new variables (components), where each sample has a score on each of those components. We can then simply create a 2-D or a 3D plot with the components on the x and y (and z) axis. We could do the same for any combination of the many components a PCoA calculates. Enough theory, let's try to put this into practice!

7.1.1 PCoA for ASV-level data with Bray-Curtis

We will illustrate what we described above using different distance measures and also applied to different taxonomic ranks. We start with a PCoA that uses a Bray Curtis matrix as input, calculated at the Genus level.

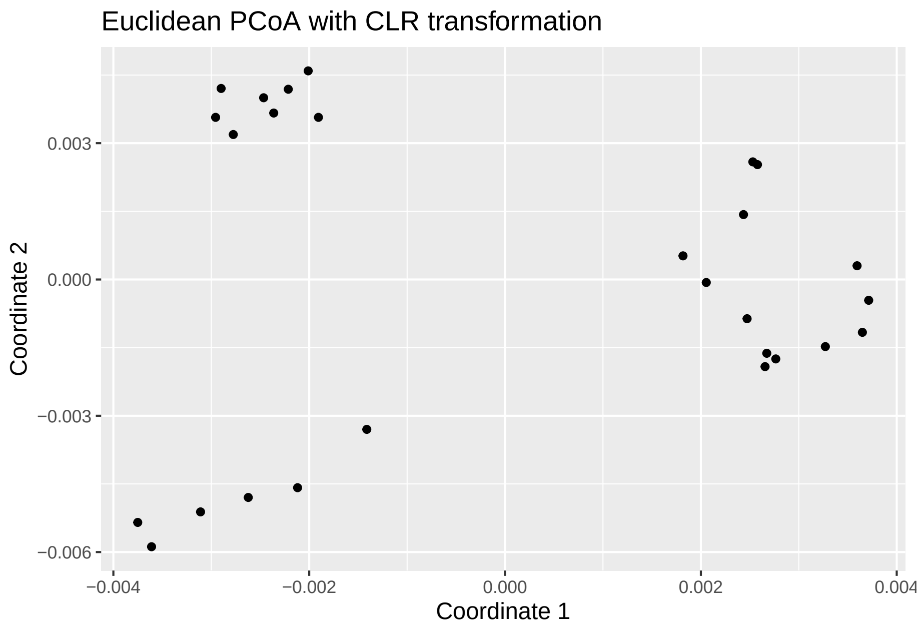
```r
# Relative abundance table
rel_abund_assay <- assays(tse)$relabundance
# Transposes it to get taxa to columns
rel_abund_assay <- t(rel_abund_assay)
# Calculates Bray-Curtis distances between samples. Because taxa is in columns,
# it is used to compare different samples.
bray_curtis_dist <- vegan::vegdist(rel_abund_assay, method = "bray")
# Does principal coordinate analysis
bray_curtis_pcoa <- ecodist::pco(bray_curtis_dist)
# all components could be found here:
# bray_curtis_pcoa$vectors
# But we only need the first two to demonstrate what we can do:
bray_curtis_pcoa_df <- data.frame(pcoa1 = bray_curtis_pcoa$vectors[,1],
                                  pcoa2 = bray_curtis_pcoa$vectors[,2])

# Creates a plot
bray_curtis_plot <- ggplot(data = bray_curtis_pcoa_df, aes(x=pcoa1, y=pcoa2)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "Bray-Curtis PCoA with relative abundances") +
  theme(title = element_text(size = 10)) # makes titles smaller

bray_curtis_plot
```



7.1.2 PCoA for ASV-level data with Aitchison distance

Now the same using Aitchison distance. This metric corresponds to Euclidean distances between CLR transformed sample abundance vectors.

```
# 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 plot
euclidean_plot <- ggplot(data = euclidean_pcoa_df, aes(x=pcoa1, y=pcoa2)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
        title = "Euclidean PCoA with CLR transformation") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_plot
```

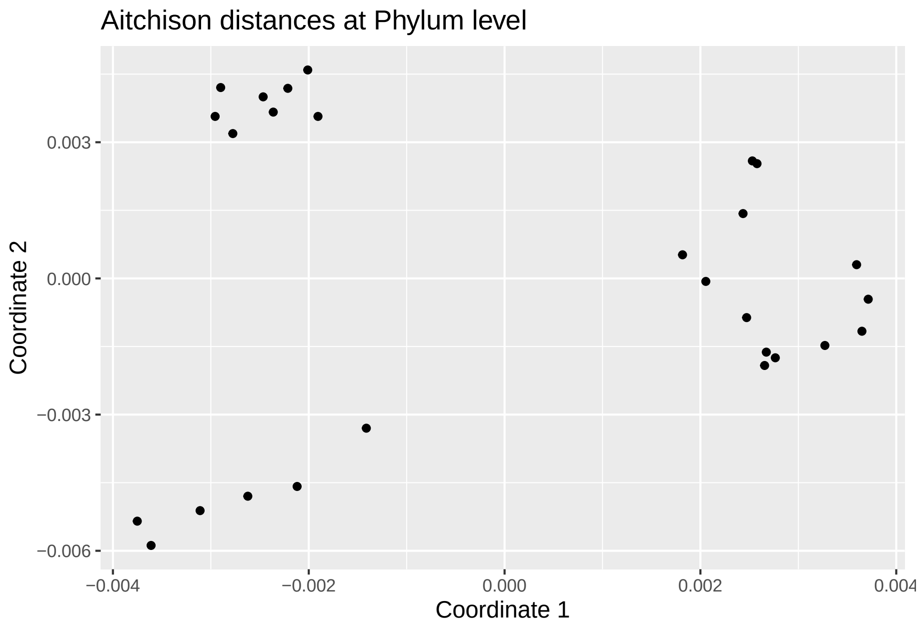


7.1.3 PCoA aggregated to Phylum level

We use again the Aitchison distances in this example but this time applied to the phylum level.

```
# Does clr transformation. Psuedocount is added, because data contains zeros.
tse_phylum <- transformCounts(tse_phylum, method = "clr", pseudocount = 1)
# Gets clr table
clr_phylum_assay <- assays(tse_phylum)$clr
# Transposes it to get taxa to columns
clr_phylum_assay <- t(clr_phylum_assay)
# Calculates Euclidean distances between samples. Because taxa is in columns,
# it is used to compare different samples.
euclidean_phylum_dist <- vegan::vegdist(clr_assay, method = "euclidean")
# Does principal coordinate analysis
euclidean_phylum_pcoa <- ecodist::pco(euclidean_phylum_dist)
# Creates a data frame from principal coordinates
euclidean_phylum_pcoa_df <- data.frame(pcoa1 = euclidean_phylum_pcoa$vectors[,1],
                                       pcoa2 = euclidean_phylum_pcoa$vectors[,2])
```

```r
# Creates a plot
euclidean_phylum_plot <- ggplot(data = euclidean_phylum_pcoa_df, aes(x=pcoa1, y=pcoa2)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "Aitchison distances at Phylum level") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_phylum_plot
```



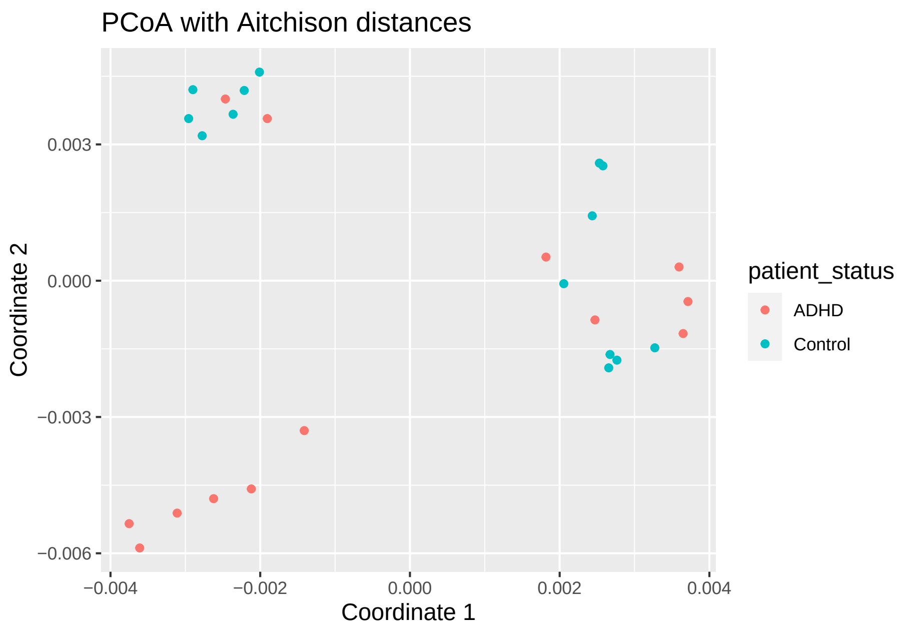
7.2 Highlighting external variables on PCoA plot

As explained in the introduction, it can help to explore the data if we map other variables on the same plot. For example, by changing the color of the points based on that variable. Let’s see how we could do that:

7.2.1 PCoA with discrete sample grouping variable shown with colors

```r
# Adds the variable we later use for coloring to the data frame
euclidean_patient_status_pcoa_df <- cbind(euclidean_pcoa_df,
                                          patient_status = colData(tse)$patient_status)
```

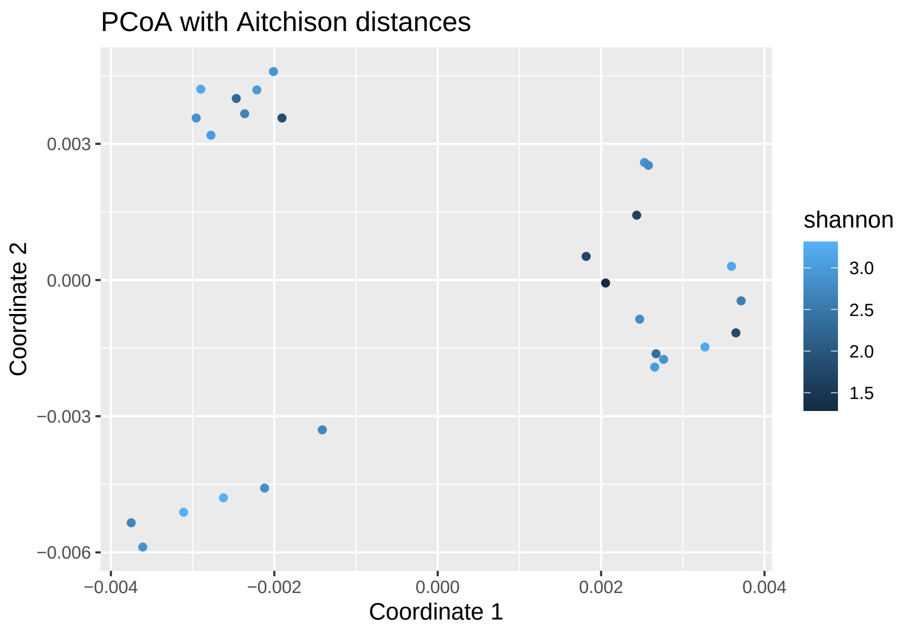
```r
# Creates a plot
euclidean_patient_status_plot <- ggplot(data = euclidean_patient_status_pcoa_df,
                                        aes(x=pcoa1, y=pcoa2,
                                            color = patient_status)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "PCoA with Aitchison distances") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_patient_status_plot
```



7.2.2 PCoA plot with continuous variable

We can do the same as above using any continuous variable. E.g. let us see how the plotted samples differ in their alpha diversity:

```r
# Adds coloring information to the data frame, creates new column
euclidean_shannon_pcoa_df <- cbind(euclidean_pcoa_df,
                                   shannon = colData(tse)$Shannon_index)

# Creates a plot
euclidean_shannon_plot <- ggplot(data = euclidean_shannon_pcoa_df,
                                 aes(x=pcoa1, y=pcoa2,
                                     color = shannon)) +
  geom_point() +
  labs(x = "Coordinate 1",
       y = "Coordinate 2",
       title = "PCoA with Aitchison distances") +
  theme(title = element_text(size = 12)) # makes titles smaller

euclidean_shannon_plot
```



7.3 Estimating associations with an external variable

Next to visualizing whether any variable is associated with differences between samples, we can also quantify the strength of the association between the variation in community composition (beta diversity) and external factors. The current standard way to do this is to perform a so-called permutational multivariate analysis of variance (PERMANOVA). This method takes as input the ASV/OTU table, which measure of distance you want to base the test on and a formula that tells the model how you think the variables are associated with each other. Let's try it:

```
# First we get the relative abundance table
rel_abund_assay <- assays(tse)$relabundance
# again transpose it to get taxa to columns
rel_abund_assay <- t(rel_abund_assay)
```

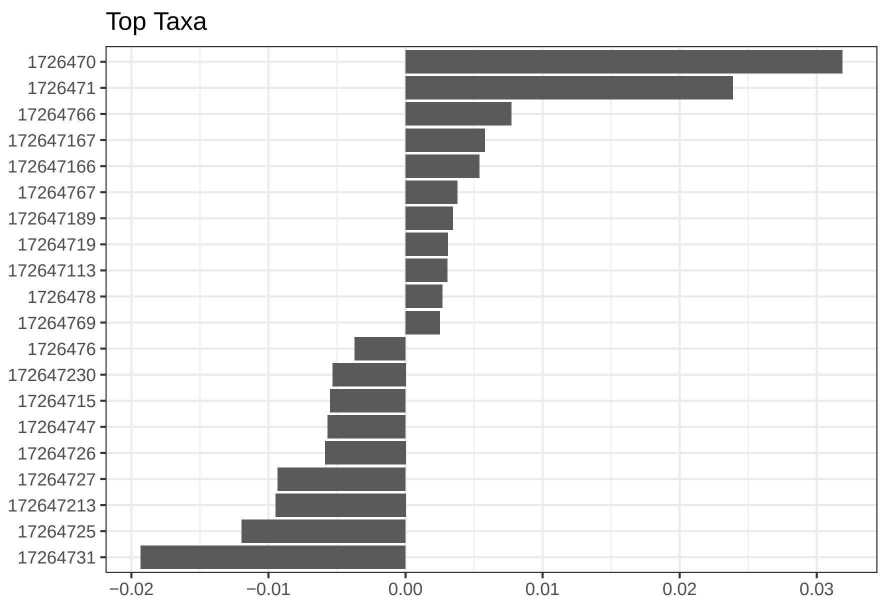
```
# then we can perform the method
permanova_cohort <- vegan::adonis(rel_abund_assay ~ cohort,
                                  data = colData(tse),
                                  permutations = 9999)

# we can obtain a the p value for our predictor:
print(paste0("Different different cohorts and variance of abundance between samples, p-
             as.data.frame(permanova_cohort$aov.tab)["cohort", "Pr(>F)"]))
```

```
## [1] "Different different cohorts and variance of abundance between samples, p-value
```

As we see, the cohort variable is not significantly associated with microbiota composition (p-value is over 0.05).

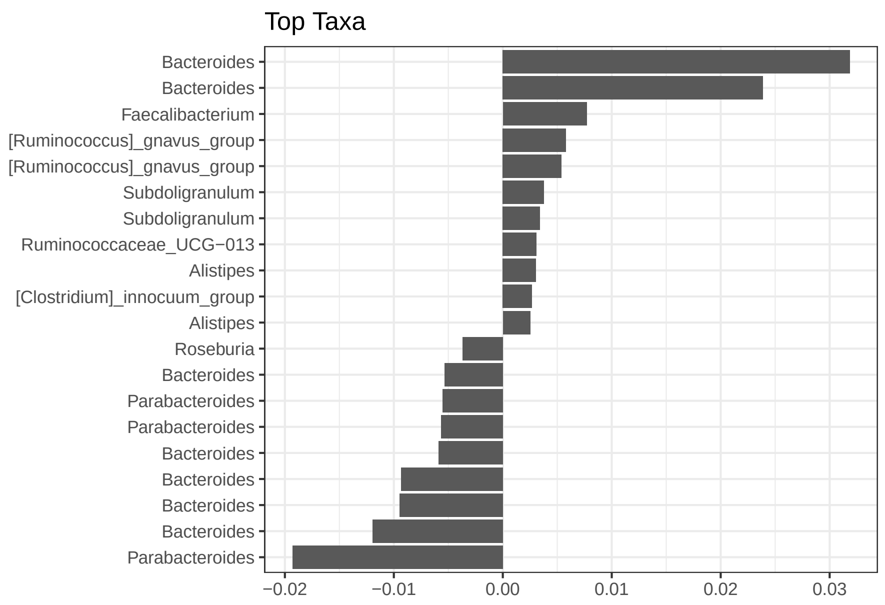
We can still visualize those taxa whose abundances are the most different between cohorts. This gives us information how taxonomic abundances tend to differ between different cohorts. In order to do that, we first need to extract the model coefficients of taxa:

```
# Gets the coefficients
coef <- coefficients(permanova_cohort)["cohort1",]
# Gets the highest coefficients
top.coef <- sort(head(coef[rev(order(abs(coef)))],20))
# Plots the coefficients
top_taxa_coeffient_plot <- 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()

top_taxa_coeffient_plot
```



The above plot shows taxa as code names, and it is hard to tell which bacterial groups they represent. However, it is easy to add human readable names. We can fetch those from our rowData. Here we use Genus level names:

```r
# Gets corresponding Genus level names and stores them to top.coef
names <- rowData(tse)[names(top.coef), ][,"Genus"]
# Adds new labels to the plot
top_taxa_coeffient_plot <- top_taxa_coeffient_plot +
  scale_y_discrete(labels = names) # Adds new labels
top_taxa_coeffient_plot
```



7.4 Further resources

For more examples, see a dedicated section on beta diversity in the online book.

Chapter 8

Community typing

8.1 Dirichlet-Multinomial Mixture Model

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.
# For this small data, takes about 10 seconds. For larger data, can take much longer
# because this demands lots of resources.
tse_dmn <- runDMN(tse, name = "DMN", k = 1:7)
```

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

```
## class: TreeSummarizedExperiment
## dim: 151 27
## metadata(1): DMN
## assays(3): counts relabundance clr
## rownames(151): 1726470 1726471 ... 17264756 17264757
## rowData names(6): Kingdom Phylum ... Family Genus
## colnames(27): A110 A12 ... A35 A38
## colData names(6): patient_status cohort ... Shannon_index Faith_diversity_index
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## rowLinks: a LinkDataFrame (151 rows)
## rowTree: 1 phylo tree(s) (151 leaves)
## colLinks: NULL
## colTree: NULL
```

Return information on metadata that the object contains.

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

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

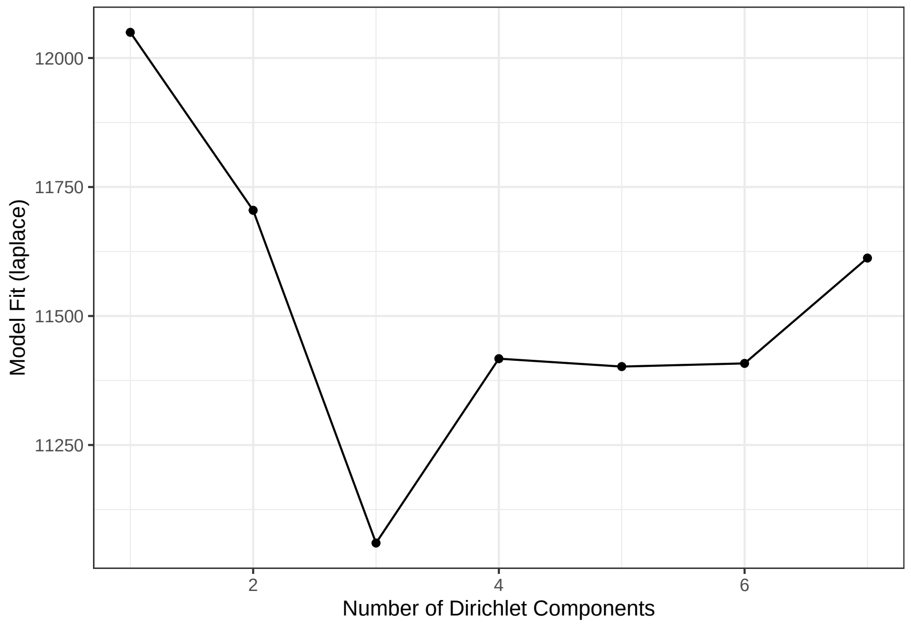
This returns a list of DMN objects for a closer investigation.

```
getDMN(tse_dmn)
```

```
## [[1]]
## class: DMN
## k: 1
## samples x taxa: 27 x 151
## Laplace: 12049.73 BIC: 12271.38 AIC: 12173.55
##
## [[2]]
## class: DMN
## k: 2
## samples x taxa: 27 x 151
## Laplace: 11704.82 BIC: 12399.15 AIC: 12202.83
##
## [[3]]
## class: DMN
## k: 3
## samples x taxa: 27 x 151
## Laplace: 11060.03 BIC: 12266.31 AIC: 11971.51
##
## [[4]]
## class: DMN
## k: 4
## samples x taxa: 27 x 151
## Laplace: 11417.3 BIC: 13047.39 AIC: 12654.11
##
## [[5]]
## class: DMN
## k: 5
## samples x taxa: 27 x 151
## Laplace: 11401.99 BIC: 13460.72 AIC: 12968.95
##
## [[6]]
## class: DMN
## k: 6
## samples x taxa: 27 x 151
## Laplace: 11408.15 BIC: 13774.56 AIC: 13184.3
##
## [[7]]
## class: DMN
## k: 7
## samples x taxa: 27 x 151
## Laplace: 11612.31 BIC: 14321.7 AIC: 13632.96
```

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

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



Return the model that has the best fit.

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

```
## class: DMN
## k: 3
## samples x taxa: 27 x 151
## Laplace: 11060.03 BIC: 12266.31 AIC: 11971.51
```

8.2 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 = "patient_status",
                               exprs_values = "counts", k = 3)

dmn_group
```

```
## class: DMNGroup
## summary:
##         k samples taxa      NLE     LogDet  Laplace      BIC      AIC
## ADHD    3      13  151 6330.860  -50.38581 5887.550 6914.386 6785.860
## Control 3      14  151 6647.269 -148.36683 6154.969 7247.655 7102.269
```

Mixture weights (rough measure of the cluster size).

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

```
##          pi    theta
## 1 0.4814815 31.27816
## 2 0.2962963 47.34448
## 3 0.2222222 92.27384
```

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

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

```
##               [,1]          [,2]          [,3]
## A110  1.000000e+00 1.256757e-144 7.560576e-205
## A12  1.013511e-116  6.148857e-93  1.000000e+00
## A15   1.000000e+00 9.617153e-119 3.401269e-234
## A19  5.556651e-112 1.828334e-107  1.000000e+00
## A21   2.212730e-93  4.754147e-96  1.000000e+00
## A23   1.000000e+00 8.868820e-111 1.936175e-161
```

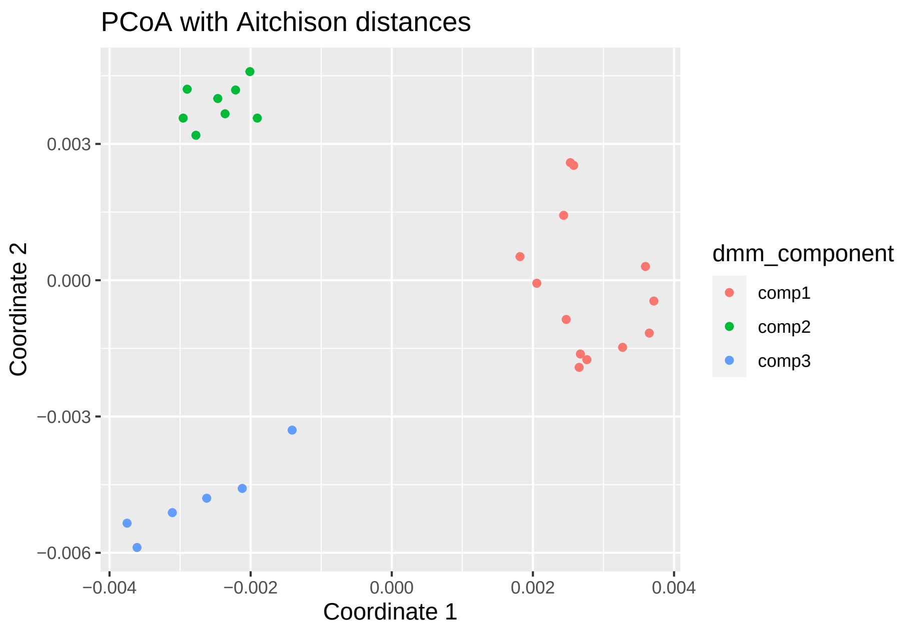
Contribution of each taxa to each component

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

```
##                 [,1]        [,2]       [,3]
## 1726470  6.352401852 2.898793384 20.1892261
## 1726471  5.288180478 0.002048022  0.1532207
## 17264731 0.001250503 9.144727789  2.0111854
## 17264726 0.140478535 1.363512443  7.5893917
## 1726472  2.104262351 3.523423613  2.6656749
## 17264724 0.072365233 0.002048022  9.8544916
```

Get the assignment probabilities

```
prob <- DirichletMultinomial::mixture(getBestDMNFit(tse_dmn))
# Add column names
colnames(prob) <- c("comp1", "comp2", "comp3")

# 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")]

# 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
```



Chapter 9

Differential abundance analysis

Here, we analyse abundances with three different methods: **Wilcoxon test**, **DESeq2**, and **ANCOM-BC**. All of these test statistical differences between groups. We analyse Genus level abundances.

9.1 Wilcoxon test

A Wilcoxon test estimates difference between two groups. It is a non-parametric alternative to a t-test, which means that the Wilcoxon test does not require normally distributed data.

Let's first collect the data for the testing purpose.

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

# Does clr transformation. Pseudocount is added, because data contains zeros, and
# clr transformation includes log transformation.
tse_genus <- transformCounts(tse_genus, method = "clr", pseudocount = 1)

# Does transpose, so samples are in rows, then creates a data frame.
abundance_analysis_data <- data.frame(t(assay(tse_genus, "clr")))

# Then we need variable for grouping samples. "patient_status" column includes information
# about patients' status. There two groups "ADHD" and "control". Let's include that to the data f
abundance_analysis_data <- cbind(abundance_analysis_data, patient_status = colData(tse_genus)$pat
```

Now we can do the Wilcoxon test. We test all the taxa by looping through columns, and store individual p-values to a vector. Then we create a data frame from collected data.

Code below does the Wilcoxon test only for columns that contain abundances, not for column that contain patient status.

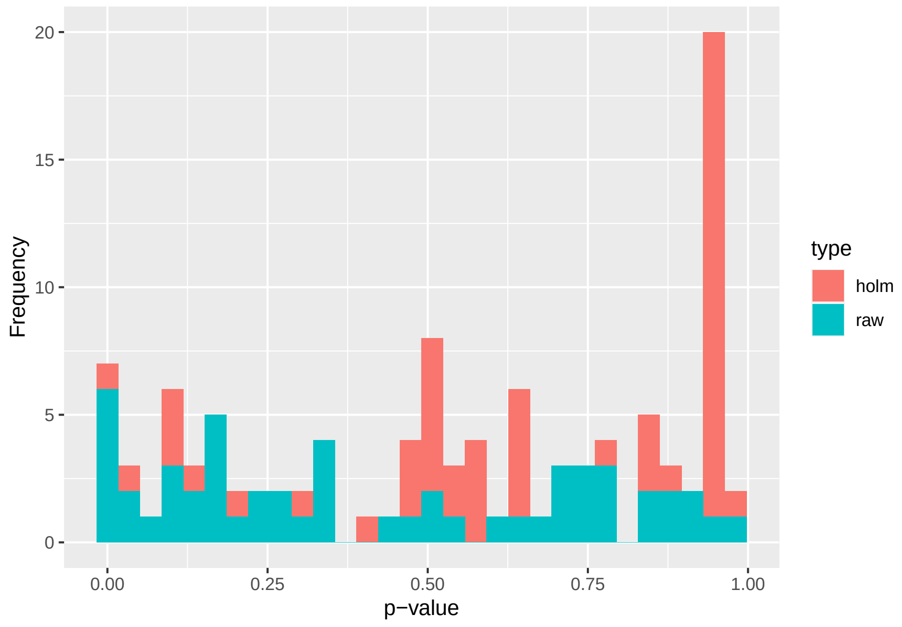
```
colnames <- names(abundance_analysis_data[, !names(abundance_analysis_data) %in% "pati

wilcoxon_p <- c() # Initialize empty vector for p-values

# Do for loop over selected column names
for (i in colnames) {

  result <- wilcox.test(abundance_analysis_data[, i] ~ patient_status,
                        data = abundance_analysis_data)

  # Stores p-value to the vector with this column name
  wilcoxon_p[[i]]  <- result$p.value

}

wilcoxon_p <- data.frame(taxa =  names(wilcoxon_p),
                         p_raw = unlist(wilcoxon_p))
```

Multiple tests were performed. These are not independent, so we need to adjust p-values for multiple testing. Otherwise, we would increase the chance of a type I error drastically depending on our p-value threshold. By applying a p-value adjustment, we can keep the false positive rate at a level that is acceptable. What is acceptable depends on our research goals. Here we use the fdr method, but there are several other methods as well.

```
wilcoxon_p$p_adjusted <- p.adjust(wilcoxon_p$p_raw, method = "fdr")
```

```
df <- data.frame(x = c(wilcoxon_p$p_raw, wilcoxon_p$p_adjusted),
                 type=rep(c("raw", "holm"),
        c(length(wilcoxon_p$p_raw),
          length(wilcoxon_p$p_adjusted))))

wilcoxon_plot <- ggplot(df) +
  geom_histogram(aes(x=x, fill=type)) +
  labs(x = "p-value", y = "Frequency")

wilcoxon_plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



9.2 DESeq2

Our second analysis method is DESeq2. This performs the data normalization automatically. It also takes care of the p-value adjustment, so we don't have to worry about that.

DESeq2 uses negative binomial distribution to detect differences in read counts between groups. Its normalization takes care of the differences between library sizes and compositions. DESeq2 analysis includes multiple steps, but they are done automatically. More information can be found, e.g., from Harvard Chan Bioinformatic Core's tutorial Introduction to DGE - ARCHIVED

Now let us show how to do this. First, run the DESeq2 analysis.

```
# Creates DESeq2 object from the data. Uses "patient_status" to create groups.
ds2 <- DESeqDataSet(tse_genus, ~patient_status)
```

```
## converting counts to integer mode
```

```
## Warning in DESeqDataSet(tse_genus, ~patient_status): 2 duplicate rownames were renamed by addi
```

```
## Warning in DESeqDataSet(tse_genus, ~patient_status): some variables in design formula are char
```

```
# Does the analysis
dds <- DESeq(ds2)
```

```
## estimating size factors
## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## -- replacing outliers and refitting for 11 genes
## -- DESeq argument 'minReplicatesForReplace' = 7
## -- original counts are preserved in counts(dds)

## estimating dispersions

## fitting model and testing
```

```
# Gets the results from the object
res <- results(dds)

# Creates a data frame from results
df <- as.data.frame(res)

# Adds taxon column that includes names of taxa
df$taxon <- rownames(df)

# Orders the rows of data frame in increasing order firstly based on column "log2FoldC
# and secondly based on "padj" column
df <- df %>% arrange(log2FoldChange, padj)

knitr::kable(head(df))
```

	baseMean	log2FoldChange	lfcSE	stat	pva
Genus:Ruminococcaceae_UCG-014	22.548297	-24.891268	2.460684	-10.115589	0.0000
Order:Bacteroidales	40.353733	-9.241798	2.136205	-4.326270	0.0000
Genus:Faecalibacterium	231.079502	-7.074433	1.745612	-4.052694	0.0000
Genus:Catabacter	18.045614	-6.615454	1.716150	-3.854823	0.0001
Genus:Butyricicoccus	2.392885	-5.179608	2.948055	-1.756957	0.0789
Order:Gastranaerophilales	2.067972	-3.054975	2.938641	-1.039588	0.2985

9.3 ANCOM-BC

The analysis of composition of microbiomes with bias correction (ANCOM-BC) is a recently developed method for differential abundance testing. It is based on an earlier published approach. This method could be recommended as part of several approaches: A recent study compared several mainstream methods and found that among another method, ANCOM-BC produced the most consistent results and is probably a conservative approach. Please note that based on this and other comparisons, no single method can be recommended across all datasets. Rather, it could be recommended to apply several methods and look at the overlap/differences.

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. 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 `tse` object to a `phyloseq` object. Then, we specify the formula. In this formula, other covariates could potentially be included to adjust for confounding. 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)
pseq_genus <- phyloseq::tax_glom(pseq, taxrank = "Genus")

out = ancombc(
  phyloseq = pseq_genus,
  formula = "patient_status",
  p_adj_method = "holm",
  zero_cut = 0.90,
  lib_cut = 0,
  group = "patient_status",
  struc_zero = TRUE,
  neg_lb = TRUE,
  tol = 1e-5,
  max_iter = 100,
  conserve = TRUE,
  alpha = 0.05,
  global = TRUE
)
res <- out$res
```

The object `out` contains all relevant information. Again, see the documentation of the function under **Value** for an explanation of all the output objects. Our question 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, we can find all differentiallt abundant taxa. Below we show the first 6 entries of this dataframe:

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

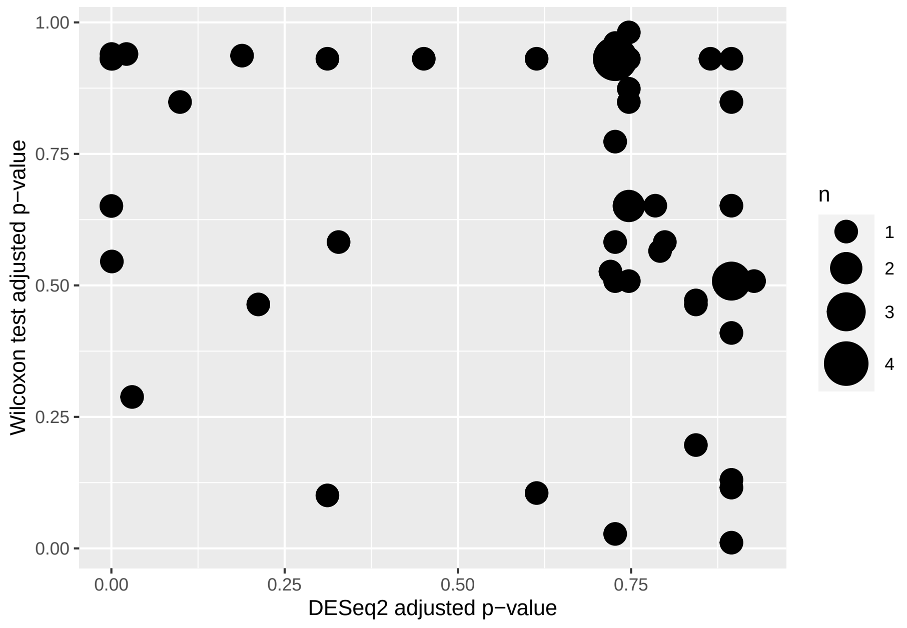
	patient_statusControl
172647198	FALSE
1726478	FALSE
172647201	FALSE
17264798	FALSE
172647195	FALSE
1726472	FALSE

In total, this method detects 13 differentially abundant taxa.

9.4 Comparison of Wilcoxon test and DESeq2

Let's compare results that we got from the Wilcoxon test and DESeq2. As we can see from the scatter plot, DESeq2 gives lower p-values than Wilcoxon test.

```
mf <- data.frame(df$padj, wilcoxon_p$p_adjusted)
p <- ggplot(mf, aes(x = df$padj, y = wilcoxon_p$p_adjusted)) +
       labs(x = "DESeq2 adjusted p-value", y = "Wilcoxon test adjusted p-value") +
       geom_count() +
 scale_size_area(max_size = 10)

print(p)
```



Prints number of p-values under 0.05

```
print(paste0("DESeq2 p-values under 0.05: ", sum(df$padj<0.05, na.rm = TRUE), "/", length(df$padj
```

```
## [1] "DESeq2 p-values under 0.05: 7/54"
```

```
print(paste0("Wilcoxon test p-values under 0.05: ", sum(wilcoxon_p$p_adjusted<0.05, na.rm = TRUE)
```

```
## [1] "Wilcoxon test p-values under 0.05: 2/54"
```

9.5 Comparison of abundance

In previous steps, we got information which taxa vary between ADHD and control groups. Let's plot those taxa in the boxplot, and compare visually if abundances of those taxa differ in ADHD and control samples. For comparison, let's plot also taxa that do not differ between ADHD and control groups.

Let's first gather data about taxa that have highest p-values.

```
# There are some taxa that do not include Genus level information. They are
# excluded from analysis.
# str_detect finds if the pattern is present in values of "taxon" column.
# Subset is taken, only those rows are included that do not include the pattern.
df <- df[ !stringr::str_detect(df$taxon, "Genus:uncultured"), ]

# Sorts p-values in decreasing order. Takes 3rd first ones. Takes those rows that match
# with p-values. Takes taxa.
highest3 <- df[df$padj %in% sort(df$padj, decreasing = TRUE)[1:3], ]$taxon

# From clr transformed table, takes only those taxa that had highest p-values
highest3 <- assay(tse_genus, "clr")[highest3, ]

# Transposes the table
highest3 <- t(highest3)

# Adds colData that includes patient status infomation
highest3 <- data.frame(highest3, as.data.frame(colData(tse_genus)))

# Some taxa names are that long that they don't fit nicely into title. So let's add there
# a line break after e.g. "Genus". Here the dot after e.g. Genus is replaced with
# ": \n"
colnames(highest3)[1:3] <- lapply(colnames(highest3)[1:3], function(x){
  # Replaces the first dot
  temp <- stringr::str_replace(x, "[.]", ": ")

  # Replace all other dots and underscores with space
  temp <- stringr::str_replace_all(temp, c("[.]" = " ", "_" = " "))
  # Adds line break so that 25 characters is the maximal width
  temp <- stringr::str_wrap(temp, width = 25)
})
```

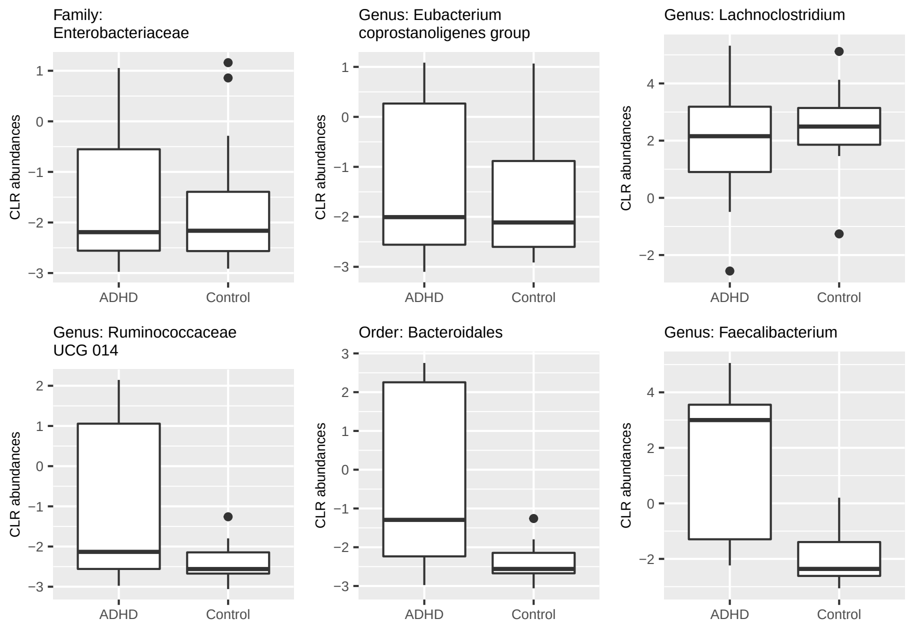
Next, let's do the same but for taxa with lowest p-values.

```
# Sorts p-values in increasing order. Takes 3rd first ones. Takes those rows that matc
# with p-values. Takes taxa.
lowest3 <- df[df$padj %in% sort(df$padj, decreasing = FALSE)[1:3], ]$taxon

# From clr transformed table, takes only those taxa that had lowest p-values
lowest3 <-assay(tse_genus, "clr")[lowest3, ]

# Transposes the table
lowest3 <- t(lowest3)

# Adds colData that includes patient status infomation
lowest3 <- data.frame(lowest3, as.data.frame(colData(tse_genus)))

# Some taxa names are that long that they don't fit nicely into title. So let's add th
# a line break after e.g. "Genus". Here the dot after e.g. Genus is replaced with
# ": \n"
colnames(lowest3)[1:3] <- lapply(colnames(lowest3)[1:3], function(x){
  # Replaces the first dot
  temp <- stringr::str_replace(x, "[.]", ": ")

  # Replace all other dots and underscores with space
  temp <- stringr::str_replace_all(temp, c("[.]" = " ", "_" = " "))

  # Adds line break so that 25 characters is the maximal width
  temp <- stringr::str_wrap(temp, width = 25)
})
```

Then we can plot these six different taxa. Let's arrange them into the same picture.

```
# Puts plots in the same picture
gridExtra::grid.arrange(

  # Plot 1
  ggplot(highest3, aes(x = patient_status, y = highest3[,1])) +
    geom_boxplot() +
    ylab("CLR abundances") + # y axis title
    ggtitle(names(highest3)[1]) + # main title
    theme(title = element_text(size = 7),
          axis.text = element_text(size = 7),
          axis.title.x=element_blank()), # makes titles smaller, removes x axis title
```

```
# Plot 2
ggplot(highest3, aes(x = patient_status, y = highest3[,2])) +
  geom_boxplot() +
  ylab("CLR abundances") + # y axis title
  ggtitle(names(highest3)[2]) + # main title
  theme(title = element_text(size = 7),
        axis.text = element_text(size = 7),
        axis.title.x=element_blank()), # makes titles smaller, removes x axis title

# Plot 3
ggplot(highest3, aes(x = patient_status, y = highest3[,3])) +
  geom_boxplot() +
  ylab("CLR abundances") + # y axis title
  ggtitle(names(highest3)[3]) + # main title
  theme(title = element_text(size = 7),
        axis.text = element_text(size = 7),
        axis.title.x=element_blank()), # makes titles smaller, removes x axis title

# Plot 4
ggplot(lowest3, aes(x = patient_status, y = lowest3[,1])) +
  geom_boxplot() +
  ylab("CLR abundances") + # y axis title
  ggtitle(names(lowest3)[1]) + # main title
  theme(title = element_text(size = 7),
        axis.text = element_text(size = 7),
        axis.title.x=element_blank()), # makes titles smaller, removes x axis title

# Plot 5
ggplot(lowest3, aes(x = patient_status, y = lowest3[,2])) +
  geom_boxplot() +
  ylab("CLR abundances") + # y axis title
  ggtitle(names(lowest3)[2]) + # main title
  theme(title = element_text(size = 7),
        axis.text = element_text(size = 7),
        axis.title.x=element_blank()), # makes titles smaller, removes x axis title

# Plot 6
ggplot(lowest3, aes(x = patient_status, y = lowest3[,3])) +
  geom_boxplot() +
  ylab("CLR abundances") + # y axis title
  ggtitle(names(lowest3)[3]) + # main title
  theme(title = element_text(size = 7),
        axis.text = element_text(size = 7),
        axis.title.x=element_blank()), # makes titles smaller, removes x axis title
  # 3 columns and 2 rows
  ncol = 3,
  nrow = 2
)
```



Chapter 10

Study material

10.1 Lecture slides

To be added.

10.2 Example solutions

To be added.

10.3 R programming resources

- 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

10.4 Resources for TreeSummarizedExperiment

- SingleCellExperiment
  - Publication
  - Project page

- SummarizedExperiment
  - Publication
  - Project page
- TreeSummarizedExperiment
  - Publication
  - Project page

10.5 Resources for phyloseq

- List of R tools for microbiome analysis
- phyloseq
- microbiome tutorial
- microbiomeutilities

10.6 Further reading

- Data Analysis and Visualization in R for Ecologists by Data Carpentry
- Modern Statistics for Modern Biology. Holmes & Huber (2018) for background in statistical analysis
- Microbiome Data Science. Shetty & Lahti, 2019

Chapter 11

Miscellaneous material

11.1 Shapiro-Wilk test

If necessary, it is possible to assess normality of the data with Shapiro-Wilk test.

```
# Does Shapiro-Wilk test. Does it only for columns that contain abundances, not for
# column that contain Groups.

normality_test_p <- c()

for (column in
     abundance_analysis_data[, !names(abundance_analysis_data) %in% "patient_status"]){
  # Does Shapiro-Wilk test
  result <- shapiro.test(column)

  # Stores p-value to vector
  normality_test_p <- c(normality_test_p, result$p.value)
}

print(paste0("P-values over 0.05: ", sum(normality_test_p>0.05), "/", length(normality_test_p)))
```

```
## [1] "P-values over 0.05: 7/54"
```

11.2 Deseq details

1. Raw counts are normalized by log-based scaling.

2. Taxa-wise variance is estimated. These values tell how much each taxa varies between samples.

3. A curve is fitted over all those taxa-wise variance estimates that we got in the last step.
   This model tells how big the variance is in a specific abundance level.
4. The model is used to shrink those individual variance estimates to avoid the effect of, e.g., small sample size and higher variance. This reduces the likelihood to get false positives.

5. Variance estimates are used to compare different groups. We receive a result that shows whether the variance is explained by groups.