**Principles of Laboratory Science (MOD008851)**

**Analysing DNA sequencing data**

**Introduction**

In this dry lab exercise we will analyse environmental DNA (eDNA) sequences from both urban (collected from a greenspace) and natural (collected from a forest) soil samples to profile the microbial communities present. DNA has been extracted from these samples and the 16S rRNA gene has been PCR amplified. The 16S rRNA genes have been DNA sequenced, and we will now analyse the produced data.

The exercise will guide you through:

* Understanding the FASTQ file format and checking sequence quality
* Processing the raw sequencing data into taxonomic units known as amplicon sequence variants (ASVs)
* Assigning taxonomy to the ASVs using a reference database (SILVA)
* Importing the data into the phyloseq R package for community analysis

This practical is designed for beginners in bioinformatics. It is self-guided you will follow the steps with provided code snippets and explanations. By the end you will have a basic workflow to go from raw DNA sequences to visualising the microbial communities in soils samples.

**Before starting**

1. Install R and RStudio (<https://posit.co/download/rstudio-desktop/>)
2. Download all the files from the following GitHub page (<https://github.com/bgregs2023/PLS>). These include the sequencing files for each sample (.fastq.gz). For each sample there are two sequencing files for the forward reads (R1.fastq.gz) and reverse reads (R2.fastq.gz). There are three samples for each condition (3 natural soils and 3 urban soils). There are also R object files (.rds) which are pre-computed results from steps that would normally take too long to run during this practical session. There is also a metadata file (metadata.csv).
3. Download the SILVA database (silva\_nr99\_v138.2\_toGenus\_trainset.fa.gz) from the following link: <https://zenodo.org/records/14169026>
4. Download the R script (**PLS.R**) from the Canvas page (<https://canvas.anglia.ac.uk/courses/45418/pages/week-6-in-person-practical-2-assessed-practical-1>)
5. In RStudio install the necessary R packages for the analysis. These include **DADA2** (for DNA sequence processing), **ShortRead** (for reading and processing FASTQ files), **ggplot2** (plotting figures), **dplyr** (for data manipulation) and **phyloseq** (for community analysis)

**Code from PLS.R**

# Install Bioconductor manager if not already installed

if (!requireNamespace("BiocManager", quietly=TRUE))

install.packages("BiocManager") # Installs BiocManager if not already present

# Install packages from Bioconductor

BiocManager::install(c("dada2", "phyloseq", "ShortRead")) # Install bioinformatics packages

# Install CRAN packages

install.packages(c("tidyverse","vegan", "ggplot2", "dplyr")) # Install data manipulation and plotting tools

# Load all the necessary packages

library(dada2) # For amplicon sequence processing

library(ShortRead) # For reading and handling FASTQ files

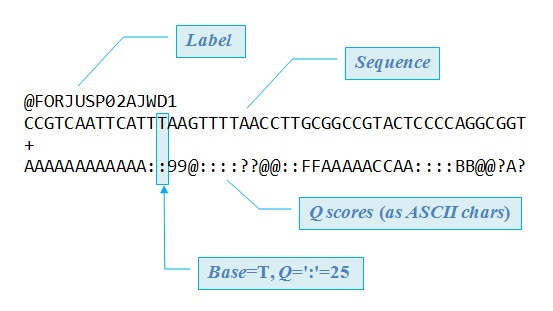
library(phyloseq) # For microbial ecology analysis and plotting

library(ggplot2) # For data visualization

library(dplyr) # For data manipulation

**Understanding FASTQ format and sequence quality**

DNA sequencing data is commonly stored in FASTQ files. A FASTQ file is a text file that contains sequencing reads and their quality scores. Each read is represented by four lines:



**Figure 1: An example of a typical FASTQ file**

Line 1: Begins with @ and contains the read identifier, a unique name for the sequencing read. May include info about the sequencer used or sample info

Line 2: The DNA sequences of the read (letters A, T, C, G and N for ambiguous bases)

Line 3: Begins with + and if often just a placeholder. It may sometimes repeat the identifier in Line 1 but is usually blank after the +

Line 4: The quality scores for each base in the sequence. Each character is encoded as an ASCII code, which corresponds to a Phred quality score which indicates the probability that a particular base (A,T,C or G) is incorrect (**see Table 1**). Higher scores mean more confidence in the base call. For example, a Phred score of 10 means a 1 in 10 chance the base may be incorrect (90% confidence), whereas a score of 40 means a chance of 1 in 10,000 the base is incorrect (99.99% confidence).

**Table 1 – Table showing the relationship between Phred quality scores (Q), their corresponding ASCII codes, and the characters used to represent then in FASTQ files. Each base call in a sequencing read is assigned a Phred score reflecting the probability that it is incorrect.**

| **Phred Score (Q)** | **ASCII Code** | **Character** |
| --- | --- | --- |
| 0 | 33 | ! |
| 1 | 34 | " |
| 2 | 35 | # |
| 3 | 36 | $ |
| 4 | 37 | % |
| 5 | 38 | & |
| 6 | 39 | ' |
| 7 | 40 | ( |
| 8 | 41 | ) |
| 9 | 42 | \* |
| 10 | 43 | + |
| 11 | 44 | , |
| 12 | 45 | - |
| 13 | 46 | . |
| 14 | 47 | / |
| 15 | 48 | 0 |
| 16 | 49 | 1 |
| 17 | 50 | 2 |
| 18 | 51 | 3 |
| 19 | 52 | 4 |
| 20 | 53 | 5 |
| 21 | 54 | 6 |
| 22 | 55 | 7 |
| 23 | 56 | 8 |
| 24 | 57 | 9 |
| 25 | 58 | : |
| 26 | 59 | ; |
| 27 | 60 | < |
| 28 | 61 | = |
| 29 | 62 | > |
| 30 | 63 | ? |
| 31 | 64 | @ |
| 32 | 65 | A |
| 33 | 66 | B |
| 34 | 67 | C |
| 35 | 68 | D |
| 36 | 69 | E |
| 37 | 70 | F |
| 38 | 71 | G |
| 39 | 72 | H |
| 40 | 73 | I |

DADA2 provides a function (**plotQualityProfile**) to visualise quality scores along reads. We’re going to check the quality scores on a subset of our FASTQ files (one forward and reverse read for the 16S genes). Use the code below to produce a quality plot from one of the natural and urban soils samples.

**Code from PLS.R**

path <- "~/Documents/PLS/16S/" # CHANGE ME to the directory containing the 16S fastq files

list.files(path) # Check the contents of the directory

# Forward and reverse fastq filenames have format

fnFs <- sort(list.files(path, pattern="\_R1.fastq.gz", full.names = TRUE)) # Forward reads

fnRs <- sort(list.files(path, pattern="\_R2.fastq.gz", full.names = TRUE)) # Reverse reads

# Plot quality profile for first natural sample (forward and reverse)

plotQualityProfile(fnFs[1]) # Visualize quality of the first forward read

plotQualityProfile(fnRs[1]) # Visualize quality of the first reverse read

# Plot quality profile for first urban sample (forward and reverse)

plotQualityProfile(fnFs[4]) # Visualize quality of a later sample (forward)

plotQualityProfile(fnRs[4]) # Visualize quality of a later sample (reverse)

A graph of quality score

AI-generated content may be incorrect.A graph of quality score

AI-generated content may be incorrect.

**Figure 2 – Example quality profile of DNA sequencing reads from our data generated using the plotQualityProfile() function in DADA2**

The quality profile plot produced by the **plotQualityProfile()** function in DADA2 is a visual summary of the base quality scores across all reads in a FASTQ file. It helps you assess whether your sequencing data are of good enough quality for downstream analysis, and where you might need to trim or filter low-quality regions.

The x-axis shows the position within the DNA sequencing read (base 1 to base 250) which allows you to see how quality changes along the length of the sequencing read. The y-axis shows Phred quality scores. A Phred score of 30 corresponds to a 1 in 1,000 chance of an incorrect base call, this is considered a high-quality base. The mean quality score at each position is shown by the green line and the quartiles of the quality score distribution is shown by the orange lines.

**Processing DNA sequencing data with DADA2**

DADA2 consists of 8 steps:

1. **Filtering and trimming:** remove low quality sequences and any that are too short or have ambiguous bases (N)
2. **Dereplication:** get rid of identical reads so you have unique sequences.
3. **Learn error rates:** estimate how likely the DNA sequencer called a base (A, T, C, G) wrong
4. **Denoising:** apply the DADA2 algorithm to work out which sequences are amplicon sequence variants (ASVs), a unique DNA sequence that represents a real organism. Works out how many ASVs there are in each sample.
5. **Merging:** merge forward and reverse DNA sequencing reads
6. **Chimera removal:** identify chimeric sequences, artifacts formed from two sequences joining during PCR and remove them.
7. **Construct a sequence table:** create a table which shows how many times an ASV is detected in each sample
8. **Assign taxonomy:** compare ASV sequences to a reference database to assign taxonomic labels (Kingdom down to Genus) for each ASV

**Filtering and trimming**

We use the **filterAndTrim** function from DADA2 to trim reads at a position where quality drops and remove reads with any ambiguous bases (N). In our data the green line doesn’t go below a quality score of 30 so we don’t need to cut the reads, and we will just filter out any short sequences (<50 bp). We use the following paramters in our code:

**maxN = 0**: removes any reads with ambiguous bases

**maxEE = c(2,2):** filters out low quality reads with more than 2 expected errors

**truncQ = 2**: trims reads at the first base with quality below 2

**minLen = 50**: discards any reads shorter than 50bp

**rm.phix = TRUE**: removes any reads contaminated with the virus PhiX which is added to the DNA sequencing run as a quality control measure

**Code from PLS.R**

# Split filenames at underscores to extract sample names

parts <- strsplit(basename(fnFs), "\_")

sample.names <- sapply(parts, function(x) paste(x[1:3], collapse = "\_")) # Create sample names

sample.names # View sample names

# Create paths for your filtered outputs to be saved on your machine

filtFs <- file.path(path, "filtered", paste0(sample.names, "\_F\_filt.fastq.gz")) # Filtered forward reads

filtRs <- file.path(path, "filtered", paste0(sample.names, "\_R\_filt.fastq.gz")) # Filtered reverse reads

names(filtFs) <- sample.names

names(filtRs) <- sample.names

# Perform filtering and trimming based on quality thresholds

out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, maxN = 0, maxEE = c(2, 2),

truncQ = 2, minLen = 50, rm.phix = TRUE, compress = TRUE, multithread = TRUE)

We would adapt the command if the quality profile looked different. In the below example the forward reads (top two; R1) are of good quality as the green line doesn’t drop below a quality score of 30 until it reaches position 240 on the x-axis. The reverse reads (bottom two; R2) are of significantly worse quality as there is a big drop at position 160 on the x-axis. In this scenario we would consider cutting the low quality reads off using DADA2

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

**Figure 3 – Example quality profile of poorer quality DNA sequencing reads generated using the plotQualityProfile() function in DADA2**

In this scenario we would run the below code with the **truncLen=c(240,160)** command to cut the forward reads at position 240 and the reverse reads at position 160.

**Example code (do not need to run this)**

# Perform filtering and trimming

out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, maxN = 0, maxEE = c(2, 2), **truncLen=c(240,160),** truncQ = 2, minLen = 50, rm.phix = TRUE, compress = TRUE, multithread = TRUE)

**Dereplication**

Dereplication condenses the data by collapsing identical reads. DADA2 uses the **derepFastq** function reads in FASTQ files and returns a dereplicated object for each sample so each unique sequence gets a unique ID with a count of how many times it was observed in that sample. We do this for all filtered files:

**Code from PLS.R**

# Dereplicate forward and reverse reads (collapse identical sequences)

derepFs <- derepFastq(filtFs)

derepRs <- derepFastq(filtRs)

names(derepFs) <- sample.names

names(derepRs) <- sample.names

**Learning error rates**

DADA2 learns the sequencing error rates from the data itself. It uses a machine-learning approach to estimate the probability of observing an incorrect base as a function of the quality score. We’ll use the **learnErrors** function on the filtered data to get an error model for forwards and reverse reads. As the sequencing files are quite large this step can take a long time to run so we will randomly subset our data to 5% in size using the **ShortRead** R package:

**Code from PLS.R**

# Define file paths to the first filtered forward and reverse reads

filt\_file\_F <- filtFs[1]

filt\_file\_R <- filtRs[1]

# Set a seed for reproducibility

set.seed(42)

# Subsample Forward Reads (5%) to speed up error learning

fq\_F <- readFastq(filt\_file\_F)

subsampled\_fq\_F <- fq\_F[sample(1:length(fq\_F), length(fq\_F) \* 0.05)]

subsampled\_path\_F <- tempfile(fileext = ".fastq.gz") # Create temporary output path

writeFastq(subsampled\_fq\_F, subsampled\_path\_F, compress = TRUE)

# Subsample Reverse Reads (5%) to speed up error learning

fq\_R <- readFastq(filt\_file\_R)

subsampled\_fq\_R <- fq\_R[sample(1:length(fq\_R), length(fq\_R) \* 0.05)]

subsampled\_path\_R <- tempfile(fileext = ".fastq.gz")

writeFastq(subsampled\_fq\_R, subsampled\_path\_R, compress = TRUE)

# Learn Error Rates from Subsampled Data

errF <- learnErrors(subsampled\_path\_F, multithread = TRUE)

errR <- learnErrors(subsampled\_path\_R, multithread = TRUE)

**Denoising – inference of ASVs**

Using the learned error rates DADA2 will process each sample’s dereplicated reads and pull out the amplicon sequence variants (ASVs) in each sample. This is done with the **dada()** function. The result of this is two dada-class objects called dadaFs and dadaRs which are lists of the ASVs and their abundances in each sample. We would run the following command:

**Example code (do not need to run this)**

#Denoising – Inference of ASVs (this will take a little while)

dadaFs <- dada(derepFs, err=errF, multithread=TRUE)

dadaRs <- dada(derepRs, err=errR, multithread=TRUE)

In this practical, we are skipping running the denoising step because it can be time-consuming, especially with high-depth sequencing data. Instead, the above command has been run in advance, and the resulting R objects have been saved as .rds files. The code below loads these precomputed objects, dadaFs for forward reads and dadaRs for reverse reads directly into your R environment using the **readRDS()** function. You can load them using a specified filename or use file.choose() to manually select the files through a file browser:

**Code from PLS.R**

# Load precomputed dada2 denoised objects to save time

dadaFs <- readRDS("dadaFs.rds")

dadaRs <- readRDS("dadaRs.rds")

# Or pick the file manually

dadaFs <- readRDS(file.choose()) # for forward reads

dadaRs <- readRDS(file.choose()) # for reverse reads

**Merging paired reads**

Since our data is paired end (DNA has been sequenced in two directions) we will now merge the forward and reverse reads for each sample to reconstruct full length sequences of the 16S rRNA gene. DADA2 does this using the **mergePairs** function which takes the dada output and dereplicated sequences and returns a list of merged sequences for each sample.

**Code from PLS.R**

# Merge paired reads based on overlap

mergers <- mergePairs(dadaFs, derepFs, dadaRs, derepRs, verbose=TRUE)

**Constructing the ASV sequence table**

Next, we combine all samples ASVs into a single amplicon sequence variant table. This table has samples as rows and ASVs as columns, and entries are the counts of each ASV in each sample. We use the **makeSequenceTable** function to build this table from the list of merged pairs. This makes an object called seqtab which includes all ASVs across all samples. Each ASV is represented by its full DNA sequence (the column name of seqtab).

**Code from PLS.R**

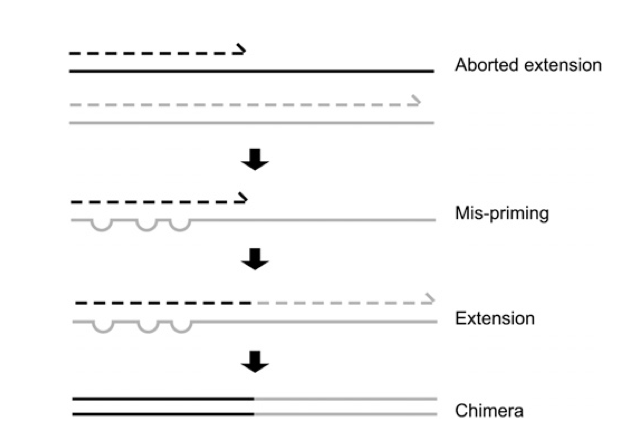
# Construct the ASV sequence table (samples as rows, ASVs as columns)

seqtab <- makeSequenceTable(mergers)

dim(seqtab) # Check dimensions of the table

**Chimera removal**

PCR chimeras occur when a DNA fragment is formed by two parent sequences joining together (**see Figure 4**). DADA2 can detect chimeras by searching for sequences in the ASV table that can be exactly reconstructed by combining two or more abundant sequences.



**Figure 4 – Formation of chimeric sequences during PCR. An aborted extension product from an earlier cycle of PCR can function as a primer in a subsequent PCR cycle. If this aborted extension product anneals a chimera can form.**

We use the **removeBimeraDenovo** function in DADA2 to remove chimeras. This function compares each ASV to every other and removes likely chimeric ones. This produces an object called seqtab.nochim which is our final ASV table with chimeras removed:

**Example code (do not need to run this)**

#Chimera removal (this will take a little while)

seqtab.nochim <- removeBimeraDenovo(seqtab, method="consensus", multithread=TRUE, verbose=TRUE)

dim(seqtab.nochim)

Since this step takes quite a long time, we will load a precomputed sequence table in which chimeric sequences have already been removed:

**Code from PLS.R**

# Load precomputed chimera-filtered table

seqtab.nochim <- readRDS("seqtab.nochim.rds")

**Assigning taxonomy**

The last step is to assign taxonomic labels to each ASV. This helps interpret the community composition (e.g. which ASV is which bacteria). We do this with the **assignTaxonomy** function in DADA2. We need a reference database of known sequences. For example, for bacteria we would use the SILVA or Greengenes database, PR2/Midori database for eukaryotes, UNITE for fungi etc. For this exercise we will use the SILVA database which includes 16S reference data (silva\_nr99\_v138.2\_toGenus\_trainset.fa.gz):

**Code from PLS.R**

# Assign taxonomy using a pre-trained SILVA classifier (can take a few minutes)

# Change the file path to where you have saved the SILVA database

taxa <- assignTaxonomy(seqtab.nochim, "~/Documents/PLS/16S/silva\_nr99\_v138.2\_toGenus\_trainset.fa.gz", multithread=TRUE)

# Load precomputed taxonomy assignments if preferred

taxa <- readRDS("taxa.rds")

taxa <- readRDS(file.choose())

This returns a matrix of assignments. Rows correspond to ASVs and columns are taxonomic ranks (Kingdom, Phylum, Class, Order, Family, Genus). DADA2 reports the highest level it can confidently assign for each ASV. If an ASV’s taxonomic assignment is ambiguous at some level you’ll see NA. To inspect the results quickly use the following code:

**Code from PLS.R**

#Inspect the results quickly

taxa\_print <- taxa

rownames(taxa\_print) <- NULL

head(taxa\_print)

**Example output**

> head(taxa\_print)

Kingdom Phylum Class Order Family Genus

[1,] "Bacteria" "Verrucomicrobiota" "Verrucomicrobiia" "Chthoniobacterales" "Chthoniobacteraceae" "Candidatus Udaeobacter"

[2,] "Bacteria" "Pseudomonadota" "Alphaproteobacteria" "Hyphomicrobiales" "Xanthobacteraceae" "Bradyrhizobium"

[3,] "Bacteria" "Pseudomonadota" "Alphaproteobacteria" "Hyphomicrobiales" "Xanthobacteraceae" NA

[4,] "Bacteria" "Pseudomonadota" "Alphaproteobacteria" "Hyphomicrobiales" "Xanthobacteraceae" "Pseudolabrys"

[5,] "Bacteria" "Pseudomonadota" "Alphaproteobacteria" "Reyranellales" "Reyranellaceae" "Reyranella"

**Microbial community analysis with phyloseq**

The phyloseq R package is used to analyse microbiome data. It can store our data (ASV table, taxonomy table and sample metadata) in a single object and provides many functions for analysing biodiversity and composition.

First, we need to import our data into phyloseq. We have three main data components:

1. ASV table (called otu\_table in phyloseq): we have the **seqtab.nochim** from DADA2
2. Taxonomy table: we have **taxa** from DADA2
3. Sample metadata: we have the **metadata.csv** file from the GitHub page which has information about each sample

Load the sample metadata into R and combine your ASV table, taxonomy table and metadata into a phyloseq object:

**Code from PLS.R**

# Read in sample metadata (CSV file with sample info)

sample\_metadata <- read.csv("~/Documents/PLS/16S/metadata.csv", row.names=1)

# Create a phyloseq object from ASV table, taxonomy, and metadata

otu <- otu\_table(seqtab.nochim, taxa\_are\_rows=FALSE) # ASVs as columns

tax <- tax\_table(taxa)

sam <- sample\_data(sample\_metadata)

ps <- phyloseq(otu, tax, sam)

ps # View summary of the phyloseq object

**Alpha diversity analysis**

Alpha diversity refers to the diversity within a single sample, such as how many bacterial species there are (richness) and how evenly distributed they are (evenness). We can use phyloseq’s **plot\_richness** function to visualise alpha diversity across samples or group (natural soils vs urban soils). The below code will plot the observed number of ASVs in each sample and Shannon diversity (which accounts for richness and eveness):

**Code from PLS.R**

# Alpha diversity: plot observed richness and Shannon diversity by Condition

plot\_richness(ps, measures=c("Observed", "Shannon"), x="Condition", color="Condition")

**Taxonomic composition visualisation**

We want to visualise the microbial community composition in our samples at both the Phylum and Genus levels. We begin by transforming the count data into relative abundances so that the values in each sample add up to 1 (i.e. 100%). We then agglomerate ASVs (group together ASVs that share the same taxonomic classification) up to the Phylum level using the **tax\_glom()** function. The resulting table is converted to a long format (a table structure where each row represents a single observation rather than having taxa as columns) using **psmelt()** which makes the data easier to plot:

We calculate the mean relative abundance of each phylum across replicate samples. To simplify the plot, we retain only the top 10 most abundant phyla and genera, and group all others into a category called “Other”

Finally, we generate a 100% stacked bar plot using ggplot2 where each bar represents the average relative abundance of phyla/genera for a given condition. Colours represent different phyla/genera and black borders are added to distinguish stacked segments.

These plots allow us to compare broad trends in microbial composition across sample types (natural vs urban soils) and help identify the dominant phyla and genera within the communities.

**Code from PLS.R**

# ---- Taxonomic Composition Plots ----

# Transform counts to relative abundance per sample

ps.rel <- transform\_sample\_counts(ps, function(x) x / sum(x))

# Agglomerate ASVs at the Phylum level

ps.phylum <- tax\_glom(ps.rel, taxrank = "Phylum")

# Convert phyloseq object to long-format dataframe (sample-taxon pairs)

ps.melt <- psmelt(ps.phylum)

# Average relative abundance by Condition and Phylum

avg\_abundance <- ps.melt %>%

group\_by(Condition, Phylum) %>%

summarise(Abundance = mean(Abundance), .groups = "drop")

# Identify top 10 phyla

top\_phyla <- avg\_abundance %>%

group\_by(Phylum) %>%

summarise(Total = sum(Abundance)) %>%

arrange(desc(Total)) %>%

slice\_head(n = 10) %>%

pull(Phylum)

# Group remaining phyla as "Other"

avg\_abundance$Phylum <- as.character(avg\_abundance$Phylum)

avg\_abundance$Phylum[!(avg\_abundance$Phylum %in% top\_phyla)] <- "Other"

avg\_abundance$Phylum <- factor(avg\_abundance$Phylum, levels = c(sort(top\_phyla), "Other"))

# Plot stacked bar chart of phylum composition by condition

ggplot(avg\_abundance, aes(x = Condition, y = Abundance, fill = Phylum)) +

geom\_bar(stat = "identity", position = "stack", color = "black") +

scale\_y\_continuous(labels = scales::percent\_format(accuracy = 1)) +

ylab("Relative Abundance (%)") +

xlab("Condition") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1),

legend.position = "right",

panel.grid = element\_blank())

# Repeat for genus-level composition

ps.genus <- tax\_glom(ps.rel, taxrank = "Genus")

ps.melt <- psmelt(ps.genus)

avg\_abundance <- ps.melt %>%

group\_by(Condition, Genus) %>%

summarise(Abundance = mean(Abundance), .groups = "drop")

top\_genera <- avg\_abundance %>%

group\_by(Genus) %>%

summarise(Total = sum(Abundance)) %>%

arrange(desc(Total)) %>%

slice\_head(n = 10) %>%

pull(Genus)

avg\_abundance$Genus <- as.character(avg\_abundance$Genus)

avg\_abundance$Genus[!(avg\_abundance$Genus %in% top\_genera)] <- "Other"

avg\_abundance$Genus <- factor(avg\_abundance$Genus, levels = c(sort(top\_genera), "Other"))

ggplot(avg\_abundance, aes(x = Condition, y = Abundance, fill = Genus)) +

geom\_bar(stat = "identity", position = "stack", color = "black") +

scale\_y\_continuous(labels = scales::percent\_format(accuracy = 1)) +

ylab("Relative Abundance (%)") +

xlab("Condition") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1),

legend.position = "right",

panel.grid = element\_blank())