16S Metabarcoding with DADA2

A step-by-step tutorial using cucumber fermentation data

Quadram Institute Bioscience

2026-05-08

Table of contents

Introduction

About this training

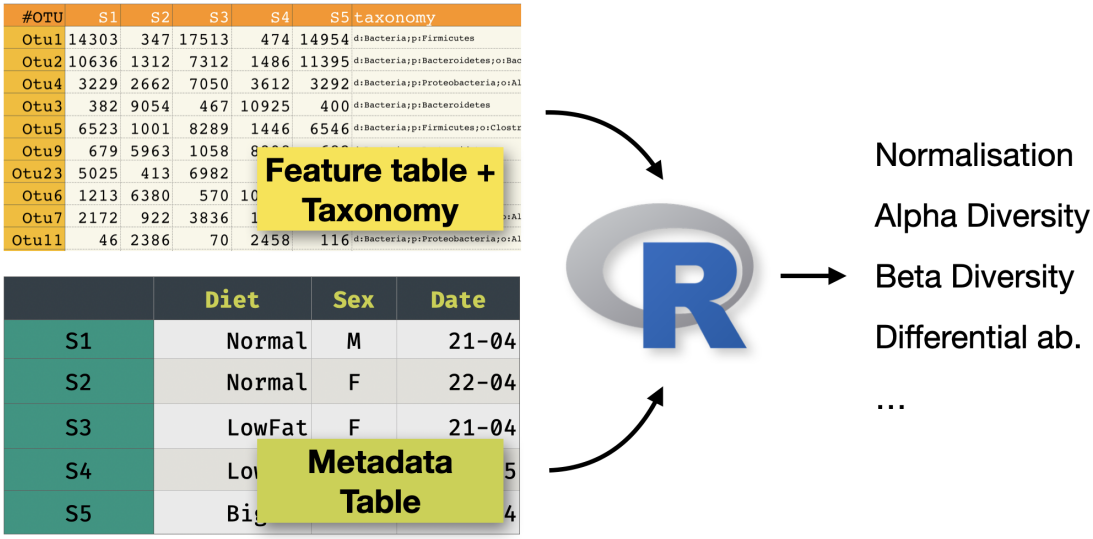


Figure 1: overview

This is a tutorial to analyse a metabarcoding dataset (in our case 16S) using DADA2. The input is a set of FASTQ files (typically, and in our case Illumina paired-end), and the output will be:

- A set of *representative sequences* (ASVs)
- A *feature counts table* (raw counts of ASVs per sample)
- The *taxonomy* of each representative sequence

We will use DADA2, and for our example will will use a recent dataset sequenced on NextSeq 2000, that uses a *binned quality* model that would not work nicely with the standard DADA2 tutorials.

> Citation: Callahan *et al.* (2026) DADA2: High-resolution sample inference from Illumina amplicon data

Trainers

This session is designed and delivered by QIB Core Bioinformatics and in particular Andrea Telatin, Judit Talas and Alise Ponsero.

What is 16S metabarcoding?

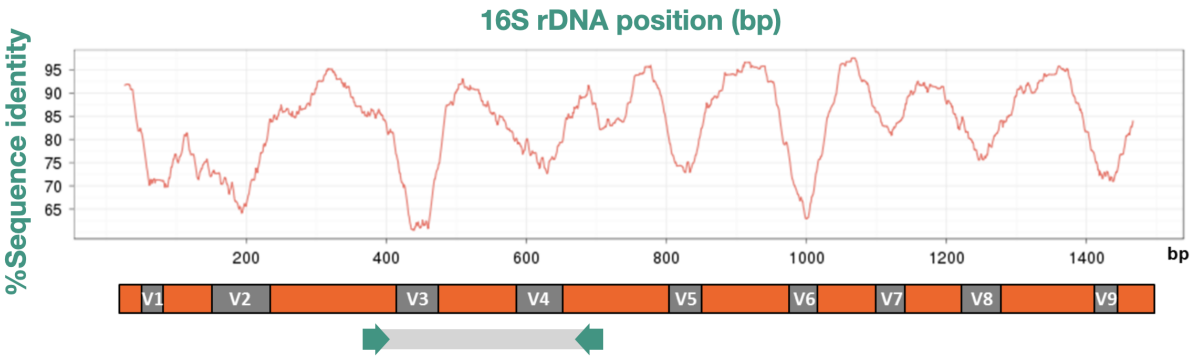


Figure 2: rDNA

16S metabarcoding is a culture-independent method to profile the bacterial communities present in a sample. It works by amplifying and sequencing a specific region of the bacterial 16S ribosomal RNA gene, a highly conserved gene found in virtually all bacteria.

A key feature of the gene is an alternation of very conserved regions (where we can design "universal primers" and hypervariable regions).

The typical workflow (*and its biases*) is:

1. Extract DNA from the sample (*are all cells breaking?*)
2. PCR-amplify the 16S gene using universal primers (*are we preferentially amplifying some taxa?*)
3. Sequence the amplicons on an Illumina platform (paired-end) (*we need to sequence only a fraction of the 1500bp gene!*)
4. Process the raw reads computationally to identify and quantify microbial taxa

ASVs vs OTUs

Historically, 16S data were analysed by clustering similar sequences into **Operational Taxonomic Units (OTUs)** at a fixed similarity threshold (typically 97%). One of the many popular packages offering OTU analysis is Mothur. This approach is fast but loses resolution, meaning that sequences that differ by even one nucleotide are collapsed together. This approach was simultaneously dealing with sequencing errors and simplyfing the identification of key taxa.

Modern pipelines instead resolve **Amplicon Sequence Variants (ASVs)**: exact, denoised sequences inferred from the data.

ASVs offer some benefits:

- Higher resolution — single-nucleotide differences are preserved
- Reproducibility — the same ASV sequence means the same biological entity across studies and platforms
- No arbitrary threshold — no need to choose a similarity cutoff (historically OTUs were clustered at 97% for no good reason)

But also some disadvantages:

- Some species will be artificially split into multiple ASVs (there is variability in 16S opersons in the same species)
- Some algorithms like DADA2 are sensitive to sequencing quality

DADA2 is the leading ASV-based pipeline and is what we will use throughout this tutorial.

The dataset

This tutorial uses real 16S amplicon data from a cucumber fermentation study. The **metadata file** allows to download the raw reads.

Three types of fermentation were compared at multiple time points:

Group	Description
Lactobacillus	Brine inoculated with a *Lactobacillus* starter culture
Spontaneous	Brine fermented without a starter culture
Commercial	A commercially produced fermented cucumber product
Controls	Negative controls (fermentation and extraction blanks)

Time points: 24 hours (`H24`), 48 hours (`H48`), 1 week (`W1`), 2 weeks (`W2`). Each condition has three replicates. In total there are **29 samples**.

Samples were sequenced on an **Illumina NextSeq 2000** — an important detail that affects how we model sequencing errors (explained in Chapter 4).

Pipeline overview

```
Raw reads (FASTQ)

Quality assessment     plotQualityProfile()

Filtering & trimming   filterAndTrim()

Error rate modelling   learnErrors()     key step for NextSeq 2000

Denoising          dada()

Merge paired reads     mergePairs()

Sequence table       makeSequenceTable()

Chimera removal       removeBimeraDenovo()
```

```
Taxonomy assignment      assignTaxonomy()

Phyloseq object        phyloseq()

Export (RDS, CSV)
```

Software requirements

All packages can be installed from Bioconductor / CRAN:

```
# Run once to install Bioconductor "installer"
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

# We can now install DADA2 and other packages
BiocManager::install(c("dada2", "phyloseq", "Biostrings"))

# And tidyverse...
install.packages(c("tidyverse"))
```

The analysis was developed and tested with:

- R 4.3
- dada2 1.28
- phyloseq 1.44
- tidyverse 2.0

R Markdown version

This website is built with Quarto and analyses a full dataset. An R markdown version that focus on a subset of the raw data is available here.

1 How to Run This Tutorial

Before diving into the analysis, you need two things: the raw sequencing reads and a working R environment with the required packages. This preliminary page covers both.

1.1 TLDR; Minimal downloads

- A subset of the reads from zenodo.org/records/20082786
- Silva138.2 Database for DADA2 from zenodo.org/records/14169026
- A script to install the required packages from GitHub
- There is an R Markdown version that was tested on the NBI HPC (source)

1.2 The dataset: Getting the raw reads

The dataset used throughout this tutorial is publicly available in the **European Nucleotide Archive (ENA)** under accession PRJEB112055.

1.2.1 Downloading via the ENA browser

1. Go to https://www.ebi.ac.uk/ena/browser/view/PRJEB112055
2. Click the **"Download"** icon and select **"FASTQ files (FTP)"** to get a TSV file listing the FTP URLs for every sample.
3. Use `wget` to download the files listed in the TSV:

```
# Download the FTP links file from ENA
wget -i <(awk -F'\t' 'NR>1 {split($7,a,";"); for(i in a) print a[i]}' filereport.tsv)
```

1.3 Running locally with RStudio

1.3.1 Install R and RStudio

1. Download and install **R** (4.3, but latest is preferred) from https://cran.r-project.org/
2. Download and install **RStudio Desktop** (free) from https://posit.co/download/rstudio-desktop/

1.3.2 Install required packages

Open RStudio and run the following once to install all dependencies:

```
# Install some tidyverse packages from CRAN
install.packages(c("tidyr","tibble","readr","ggplot2","patchwork"))

# Install the Bioconductor package manager
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

# Install DADA2 and phyloseq from Bioconductor
BiocManager::install(c("dada2", "phyloseq", "Biostrings"))
```

Installation can take several minutes the first time. Once complete, verify everything loaded correctly:

```
library(dada2)
```

> **Version check**
>
> The tutorial was developed with dada2 1.28, phyloseq 1.44, and tidyverse 2.0. If you see unexpected errors, check `packageVersion("dada2")` and update via `BiocManager::install("dada2")` if needed.

1.4 Running on the NBI HPC

> **HPC account required**
>
> You need an active NBI HPC account to follow this route. If you do not have one, contact your line manager or the NBI IT helpdesk to request access.

The NBI HPC provides RStudio through the **Open on Demand** web portal — no command-line setup required.

1.4.1 Step-by-step

1. Open your browser and go to https://ood.hpc.nbi.ac.uk/
2. Log in with your NBI HPC credentials.
3. In the top navigation bar click **"Interactive Apps"** and select **RStudio**.
4. Choose the compute resources you need (a 8 cores and 64 GB RAM are enough for this tutorial) and click **"Launch"**.
5. Wait for the job to start, then click **"Connect to RStudio Server"**.
6. Run the installation script to install dependencies

1.4.2 Installing packages on the HPC

The shared R installation on the HPC does **not** include DADA2, phyloseq, or tidyverse by default. Run the same installation commands as for a local setup — R will install the packages into your personal library (e.g. `~/r/lib/`):

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install(c("dada2", "phyloseq", "Biostrings"))

install.packages("tidyverse")
```

2 Setup and Quality Assessment

2.1 Load libraries

We start by loading the packages needed for the entire analysis. It is good practice to load all dependencies at the top of the script so that any missing package is caught immediately.

```
library(dada2)      # Core DADA2 functions
library(tidyverse)  # Data manipulation and ggplot2
library(phyloseq)   # Microbiome data structures (used later)
```

2.2 Configure paths

We define all file paths relative to this project's working directory. Using relative paths makes the analysis portable — moving the project folder will not break the code.

```
# Directory containing the renamed, trimmed FASTQ files
# (reads are already primer-trimmed and renamed to clean sample names)
path_reads <- "../reads/"

# SILVA reference database for taxonomy assignment (used in Chapter 5)
path_silva <- "../db/silva_nr_v138_train_set.fa.gz"

# Directory for intermediate outputs
path_filtered <- "filtered"
path_precomp  <- "precomputed"

dir.create(path_filtered, showWarnings = FALSE)
dir.create(path_precomp,  showWarnings = FALSE)
```

2.3 Removing primers

DADA2 works on raw reads but after primer removal. We used cutadapt to remove the primers.

- Primer forward: `CCTACGGGNGGCWGCAG`
- Primer reverse: `GGACTACHVGGGTATCTAATCC`

The typical command to remove primers using cutadapt is:

```
cutadapt -a FWDPRIMER...RCREVPRIMER -A REVPRIMER...RCFWDPRIMER \
   --discard-untrimmed -m 100:100 -j 8 \
   -o sample1_noprimer_R1.fq.gz -p sample1_noprimer_R2.fq.gz \
   sample1_R1.fq.gz sample1_R2.fq.gz
```

- `-a` and `-A` provide the primers to trim, not the three dots `...` to separate the pairs. The first is applied to the first pair, while the second in the second pair. We provide *both* primers to each end for the case when the amplicon is shorther than the read length (e.g. ITS sequencing)
- `--discard-untrimmed` means not to save reads where the primers were not detected at all
- `-m X:Y` means not to save reads shorter than X (in R1) and Y (in R2), by default we could end up with empty reads
- `-o R1` and `-p R2` will specify the output files
- `-j INT` is for multithreading (in the example 8 cores used)
- the input are the positional arguments at the end (forward and reverse respectively)

2.4 Discover input files

DADA2 processes forward (R1) and reverse (R2) reads separately until the merging step. We find all FASTQ files matching each pattern and sort them to guarantee that the forward file at position `i` corresponds to the reverse file at position `i`.

```
fnFs <- sort(list.files(path_reads, pattern = "_R1.fastq.gz", full.names = TRUE))
fnRs <- sort(list.files(path_reads, pattern = "_R2.fastq.gz", full.names = TRUE))

cat("Forward read files found:", length(fnFs), "\n")
```

```
Forward read files found: 29
```

```
cat("Reverse read files found:", length(fnRs), "\n")
```

```
Reverse read files found: 29
```

2.4.1 Extract sample names

Sample names are extracted from the forward file names. Our files follow the naming convention `<SampleID>_R1.fastq.gz`, so we strip the `_R1.fastq.gz` suffix.

```
sample.names <- sub("_R1\\.fastq\\.gz$", "", basename(fnFs))
head(sample.names, 8)
```

```
[1] "Commercial_R1"        "Commercial_R2"        "Commercial_R3"
[4] "Control_extraction"   "Control_fermentation" "Lactobacillus_H24_R1"
[7] "Lactobacillus_H24_R2" "Lactobacillus_H24_R3"
```

> **File naming matters**
>
> DADA2 assumes the forward and reverse file vectors are in the same order. Using `sort()` on both ensures correct pairing even when the filesystem returns files in an unexpected order.

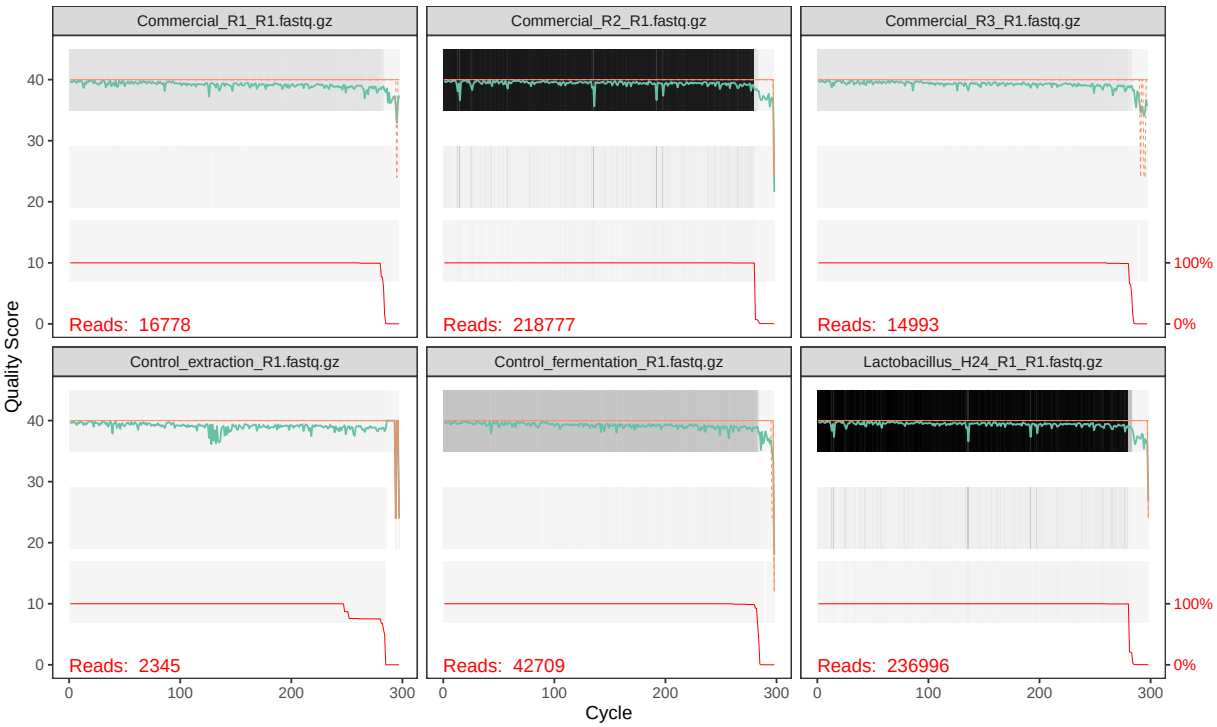
2.5 Quality assessment

Before processing reads it is essential to visualise their quality profiles. This informs the truncation lengths we will use in the next chapter.

> Quick preview online
>
> you can check the quality profile using our utility website, without loading R at all.

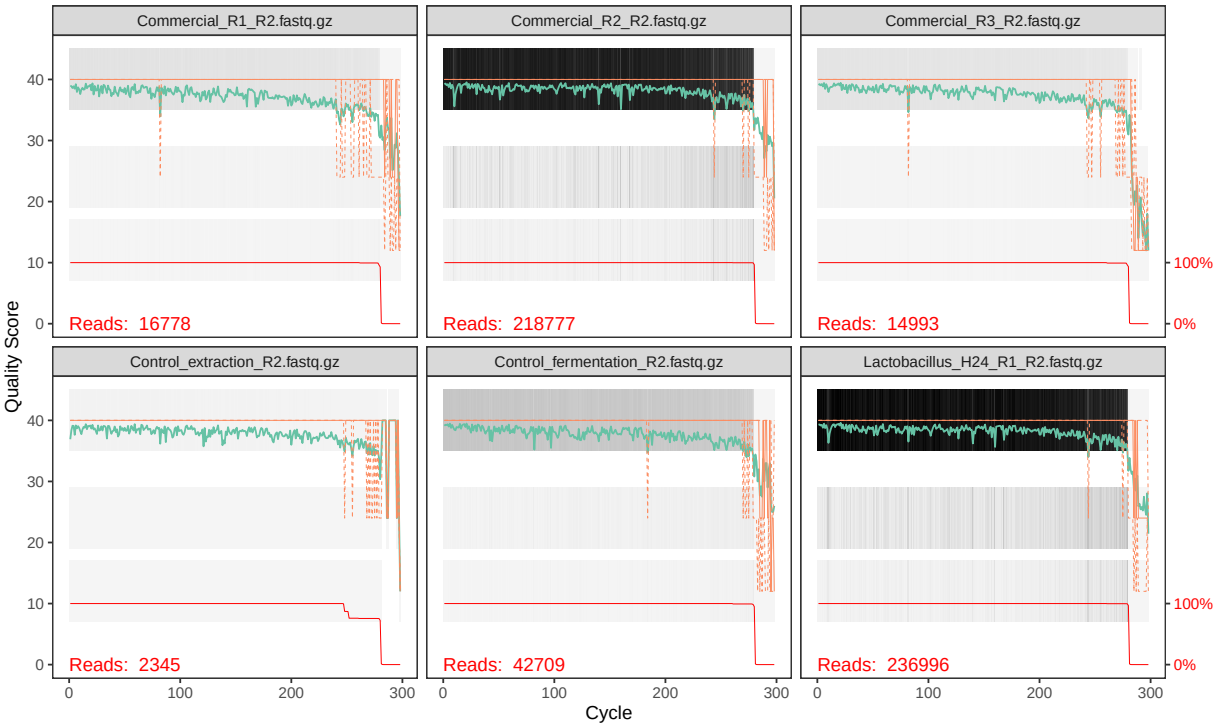
2.5.1 Forward reads

```
plotQualityProfile(fnFs[1:6])
```



2.5.2 Reverse reads

```
plotQualityProfile(fnRs[1:6])
```



2.6 Interpreting quality profiles

Each panel shows one sample. The *green line* is the mean quality score at each position, the *orange lines* are the quartiles, and the *red line* (at the bottom) shows the proportion of reads that extend to that position.

What to look for:

- **Drop in mean quality**: reads typically degrade towards the 3 end. Look for where the green line consistently falls below Q30.
- **Reverse reads are usually worse** than forward reads — this is normal for paired-end sequencing.
- **Truncation point**: you need to cut reads short enough to remove low-quality bases, but long enough so that the forward and reverse reads still overlap when merged.

Note that we also need to evaluate if we will be able to merge the two reads (if this is intended to happen). A typical V3-V4 amplicon is 460bp long, which means that 300bp paired end sequencing leaves a nice overlap to allow merging. If we need to trim a lot of bases so that their sum is going to be less than ~490bp, the overlap might not be sufficient.

2.6.1 Quality patterns in NextSeq 2000 data

This dataset was sequenced on an **Illumina NextSeq 2000**, which uses a 2-colour chemistry and reports **binned** (discrete) quality scores rather than the continuous Phred scores produced by older instruments like MiSeq. You will typically see quality scores clustering at a small number of levels (e.g. Q2, Q12, Q23, Q37) rather than spreading smoothly from 0 to 40.

This binning has an important consequence for DADA2's error modelling step — we will address it in Chapter 4.

Choosing truncation lengths

Based on these profiles, we will truncate:

- **Forward reads** at position **260** (quality remains high to this point)
- **Reverse reads** at position **250** (slightly shorter to remove end degradation)

After truncation, the overlap between the two reads is sufficient for merging the V3–V4 amplicon (~460 bp expected product; reads of 260 + 250 = 510 bp total gives ~50 bp overlap, which is adequate).

3 Filtering and Trimming

3.1 Why filter reads?

Raw sequencing reads contain:

- **Low-quality bases** at the 3 end that increase error rates in denoising
- **Ambiguous bases** (N) that DADA2 cannot handle
- **PhiX spike-in reads** — a control used for calibration during sequencing
- Occasionally, reads contaminated by the **PhiX bacteriophage genome**

The `filterAndTrim()` function removes all of these in a single step and also truncates reads to a fixed length, which is required for consistent error modelling.

3.2 Define filtered file paths

Here is where we will *save* the filtered reads.

```
filtFs <- file.path(path_filtered,
                    paste0(sample.names, "_F_filt.fastq.gz"))
filtRs <- file.path(path_filtered,
                    paste0(sample.names, "_R_filt.fastq.gz"))

# Give each vector element the corresponding sample name for easy indexing later
names(filtFs) <- sample.names
names(filtRs) <- sample.names
```

3.3 Run filterAndTrim

The *filterAndTrim* function allows to filter our reads that should be already free of sequencing primers!

```
out <- filterAndTrim(
  fnFs, filtFs,
  fnRs, filtRs,
  truncLen = c(260, 250),   # truncate F reads at 260 bp, R reads at 250 bp
  maxN     = 0,             # discard any read with an ambiguous base
  maxEE    = c(2, 2),       # maximum expected errors per read (F and R separately)
  truncQ   = 10,            # truncate at first base with quality  10
  rm.phix  = TRUE,          # remove reads matching PhiX genome
  compress = TRUE,          # write gzip-compressed output
  multithread = TRUE        # use all available CPU cores
```

```
)
saveRDS(out, file.path(path_precomp, "out.rds"))
```

3.4 Parameter guide

Parameter	Value	Rationale
`truncLen`	`c(260, 250)`	Keeps high-quality region; must be long enough for reads to overlap after merging
`maxN`	`0`	DADA2 requires zero ambiguous bases
`maxEE`	`c(2, 2)`	Maximum *expected errors* (sum of error probabilities along the read); 2 is a standard threshold
`truncQ`	`10`	Removes the very-low-quality tail before the maxEE calculation
`rm.phix`	`TRUE`	Always advisable; PhiX contamination is common in Illumina data

What is Expected Errors (EE)?

The **expected errors** metric (introduced by Edgar & Flyvbjerg 2015) is more informative than a simple minimum quality threshold. For a read with quality scores Q , Q , …, Q , the EE is:

$$EE = \sum_{i=1}^{n} 10^{-Q_i/10}$$

A read with `maxEE = 2` can have no more than 2 expected sequencing errors in total. This is more lenient than requiring every base to exceed Q30, while still keeping read quality high.

3.5 Inspect filtering statistics

```
# Convert to data frame and add sample names
filter_stats <- as.data.frame(out)
filter_stats$sample   <- rownames(filter_stats)
filter_stats$pct_kept <- round(100 * filter_stats$reads.out / filter_stats$reads.in, 1)

filter_stats[, c("sample", "reads.in", "reads.out", "pct_kept")]
```

```
                                                sample reads.in
Commercial_R1_R1.fastq.gz             Commercial_R1_R1.fastq.gz    16778
Commercial_R2_R1.fastq.gz             Commercial_R2_R1.fastq.gz   218777
```

```
Commercial_R3_R1.fastq.gz                Commercial_R3_R1.fastq.gz    14993
Control_extraction_R1.fastq.gz     Control_extraction_R1.fastq.gz     2345
Control_fermentation_R1.fastq.gz Control_fermentation_R1.fastq.gz    42709
Lactobacillus_H24_R1_R1.fastq.gz Lactobacillus_H24_R1_R1.fastq.gz   236996
Lactobacillus_H24_R2_R1.fastq.gz Lactobacillus_H24_R2_R1.fastq.gz   271744
Lactobacillus_H24_R3_R1.fastq.gz Lactobacillus_H24_R3_R1.fastq.gz   185402
Lactobacillus_H48_R1_R1.fastq.gz Lactobacillus_H48_R1_R1.fastq.gz   232087
Lactobacillus_H48_R2_R1.fastq.gz Lactobacillus_H48_R2_R1.fastq.gz   277388
Lactobacillus_H48_R3_R1.fastq.gz Lactobacillus_H48_R3_R1.fastq.gz   257945
Lactobacillus_W1_R1_R1.fastq.gz   Lactobacillus_W1_R1_R1.fastq.gz   237242
Lactobacillus_W1_R2_R1.fastq.gz   Lactobacillus_W1_R2_R1.fastq.gz   283367
Lactobacillus_W1_R3_R1.fastq.gz   Lactobacillus_W1_R3_R1.fastq.gz   135637
Lactobacillus_W2_R1_R1.fastq.gz   Lactobacillus_W2_R1_R1.fastq.gz   270194
Lactobacillus_W2_R2_R1.fastq.gz   Lactobacillus_W2_R2_R1.fastq.gz   262219
Lactobacillus_W2_R3_R1.fastq.gz   Lactobacillus_W2_R3_R1.fastq.gz   280448
Spontaneous_H24_R1_R1.fastq.gz     Spontaneous_H24_R1_R1.fastq.gz   260410
Spontaneous_H24_R2_R1.fastq.gz     Spontaneous_H24_R2_R1.fastq.gz   297828
Spontaneous_H24_R3_R1.fastq.gz     Spontaneous_H24_R3_R1.fastq.gz   258467
Spontaneous_H48_R1_R1.fastq.gz     Spontaneous_H48_R1_R1.fastq.gz   311391
Spontaneous_H48_R2_R1.fastq.gz     Spontaneous_H48_R2_R1.fastq.gz   302367
Spontaneous_H48_R3_R1.fastq.gz     Spontaneous_H48_R3_R1.fastq.gz   295072
Spontaneous_W1_R1_R1.fastq.gz       Spontaneous_W1_R1_R1.fastq.gz   266958
Spontaneous_W1_R2_R1.fastq.gz       Spontaneous_W1_R2_R1.fastq.gz   271112
Spontaneous_W1_R3_R1.fastq.gz       Spontaneous_W1_R3_R1.fastq.gz   295075
Spontaneous_W2_R1_R1.fastq.gz       Spontaneous_W2_R1_R1.fastq.gz   291669
Spontaneous_W2_R2_R1.fastq.gz       Spontaneous_W2_R2_R1.fastq.gz   259494
Spontaneous_W2_R3_R1.fastq.gz       Spontaneous_W2_R3_R1.fastq.gz   314666
                            reads.out pct_kept
Commercial_R1_R1.fastq.gz           13226     78.8
Commercial_R2_R1.fastq.gz          178641     81.7
Commercial_R3_R1.fastq.gz           12109     80.8
Control_extraction_R1.fastq.gz       1419     60.5
Control_fermentation_R1.fastq.gz    34239     80.2
Lactobacillus_H24_R1_R1.fastq.gz   193289     81.6
Lactobacillus_H24_R2_R1.fastq.gz   221122     81.4
Lactobacillus_H24_R3_R1.fastq.gz   150467     81.2
Lactobacillus_H48_R1_R1.fastq.gz   189922     81.8
Lactobacillus_H48_R2_R1.fastq.gz   227640     82.1
Lactobacillus_H48_R3_R1.fastq.gz   210626     81.7
Lactobacillus_W1_R1_R1.fastq.gz    191040     80.5
Lactobacillus_W1_R2_R1.fastq.gz    233210     82.3
Lactobacillus_W1_R3_R1.fastq.gz    107509     79.3
Lactobacillus_W2_R1_R1.fastq.gz    219755     81.3
Lactobacillus_W2_R2_R1.fastq.gz    213873     81.6
Lactobacillus_W2_R3_R1.fastq.gz    230240     82.1
Spontaneous_H24_R1_R1.fastq.gz     209712     80.5
Spontaneous_H24_R2_R1.fastq.gz     241630     81.1
Spontaneous_H24_R3_R1.fastq.gz     209072     80.9
Spontaneous_H48_R1_R1.fastq.gz     253245     81.3
Spontaneous_H48_R2_R1.fastq.gz     243119     80.4
Spontaneous_H48_R3_R1.fastq.gz     238579     80.9
```

```
Spontaneous_W1_R1_R1.fastq.gz       217397     81.4
Spontaneous_W1_R2_R1.fastq.gz       218668     80.7
Spontaneous_W1_R3_R1.fastq.gz       239236     81.1
Spontaneous_W2_R1_R1.fastq.gz       237723     81.5
Spontaneous_W2_R2_R1.fastq.gz       208982     80.5
Spontaneous_W2_R3_R1.fastq.gz       256480     81.5
```

```
cat("Median reads retained:", median(filter_stats$pct_kept), "%\n")
```

```
Median reads retained: 81.2 %
```

```
cat("Minimum reads retained:", min(filter_stats$pct_kept), "%\n")
```

```
Minimum reads retained: 60.5 %
```

A retention rate above **70–80 %** is generally acceptable. If many samples lose more than 30 % of reads at this step, consider relaxing `maxEE` slightly (e.g. to 5) or revisiting the primer-trimming step upstream.

3.6 Check filtered files exist

A small sanity check: some samples may have lost all reads (e.g. negative controls with very low input). DADA2 will error if it tries to open an empty or absent file.

```
# Keep only samples for which filtered files were actually created
exists_F <- file.exists(filtFs)
exists_R <- file.exists(filtRs)

if (any(!exists_F | !exists_R)) {
  dropped <- sample.names[!exists_F | !exists_R]
  message("The following samples had no reads after filtering and will be excluded: ",
          paste(dropped, collapse = ", "))
  filtFs <- filtFs[exists_F & exists_R]
  filtRs <- filtRs[exists_F & exists_R]
  sample.names <- names(filtFs)
}

cat("Samples remaining:", length(sample.names), "\n")
```

```
Samples remaining: 29
```

4 Error Rate Modelling

4.1 What are error rates?

Every sequencing instrument makes mistakes. A base may be called as A when it was actually C. The probability of each such substitution depends on the quality score reported for that position: a base called at Q30 has a 1-in-1000 chance of being wrong, while a Q20 base has a 1-in-100 chance.

DADA2's denoising algorithm uses a **parametric error model** — learned from the data itself — to estimate the true frequency of each type of error at each quality score. This model is then used to distinguish genuine biological variants from PCR/sequencing errors.

4.2 The problem with binned quality scores

The standard DADA2 error estimation assumes that error rates vary continuously and monotonically with the reported quality score: higher quality → lower error rate. This assumption holds well for **MiSeq** data.

NextSeq 2000 and NovaSeq use a 2-colour chemistry that reports quality scores in a small number of discrete levels (typically Q2, Q12, Q23, Q37). This **binning** creates a mismatch: many bases with very different true error rates share the same quality label, and some quality levels may have zero or very few observations. The default `loessErrfun` fails to fit a smooth, monotone curve through such sparse, clustered data.

The solution is to use a modified loess fitting function that:

1. Adds a pseudocount to avoid log(0) issues
2. Adjusts the loess bandwidth and weighting
3. Enforces monotonicity explicitly after fitting

We provide four variants (Options 1–4). **Option 4** is recommended for NextSeq 2000 and NovaSeq data and is the one used in this tutorial.

4.3 The four error model variants

All four variants share the same overall structure. The differences are in how `loess()` is called:

Option	Span	Degree	Weights	Best for
1	2 (very wide)	default (2)	log10(tot)	Heavily binned data
2	default	default (2)	tot (counts)	Standard MiSeq-like
3	default	default (2)	log10(tot)	Intermediate
4	**0.95**	**1**	**log10(tot)**	**NextSeq / NovaSeq**

Option 4 uses `degree = 1` (locally linear rather than locally quadratic) and a wide `span = 0.95`, which prevents over-fitting to the sparse quality bins and produces smooth, monotone error curves.

4.4 Define the error model functions

```
# Option 4: recommended for NextSeq 2000 / NovaSeq binned quality scores
loessErrfun_mod4 <- function(trans) {
  qq  <- as.numeric(colnames(trans))
  est <- matrix(0, nrow = 0, ncol = length(qq))

  for (nti in c("A", "C", "G", "T")) {
    for (ntj in c("A", "C", "G", "T")) {
      if (nti != ntj) {
        errs  <- trans[paste0(nti, "2", ntj), ]
        tot   <- colSums(trans[paste0(nti, "2", c("A", "C", "G", "T")), ])
        rlogp <- log10((errs + 1) / tot)   # +1 pseudocount
        rlogp[is.infinite(rlogp)] <- NA

        df <- data.frame(q = qq, errs = errs, tot = tot, rlogp = rlogp)

        # Locally-linear loess with log-count weights - the key change for binned data
        mod.lo <- loess(rlogp ~ q, df,
                        weights = log10(tot),
                        degree  = 1,
                        span    = 0.95)

        pred <- predict(mod.lo, qq)
        # Extend predictions to quality levels with no observations
        maxrli <- max(which(!is.na(pred)))
        minrli <- min(which(!is.na(pred)))
        pred[seq_along(pred) > maxrli] <- pred[[maxrli]]
        pred[seq_along(pred) < minrli] <- pred[[minrli]]
        est <- rbind(est, 10^pred)
      }
    }
  }

  # Clamp to a sensible range
  MAX_ERROR_RATE <- 0.25
  MIN_ERROR_RATE <- 1e-7
  est[est > MAX_ERROR_RATE] <- MAX_ERROR_RATE
  est[est < MIN_ERROR_RATE] <- MIN_ERROR_RATE

  # Enforce monotonicity: error rate at quality X must be   error rate at quality 40
  estorig <- est
  est <- est %>%
    data.frame() %>%
    mutate(across(everything(), ~ if_else(. < X40, X40, .))) %>%
```

```
    as.matrix()
  rownames(est) <- rownames(estorig)
  colnames(est) <- colnames(estorig)

  # Build the full 4×4 error matrix (including self-transitions)
  err <- rbind(
    1 - colSums(est[1:3, ]),  est[1:3, ],
    est[4, ],  1 - colSums(est[4:6, ]),  est[5:6, ],
    est[7:8, ], 1 - colSums(est[7:9, ]), est[9, ],
    est[10:12, ], 1 - colSums(est[10:12, ])
  )
  rownames(err) <- paste0(rep(c("A", "C", "G", "T"), each = 4), "2",
                          c("A", "C", "G", "T"))
  colnames(err) <- colnames(trans)
  return(err)
}
```

ℹ Options 1, 2, and 3 (click to expand)

These are provided for comparison. If you are processing MiSeq data you may find Option 2 (standard weights) or Option 3 (log-weight without degree/span change) more appropriate.

```
loessErrfun_mod1 <- function(trans) {
  qq  <- as.numeric(colnames(trans))
  est <- matrix(0, nrow = 0, ncol = length(qq))
  for (nti in c("A","C","G","T")) {
    for (ntj in c("A","C","G","T")) {
      if (nti != ntj) {
        errs  <- trans[paste0(nti,"2",ntj),]
        tot   <- colSums(trans[paste0(nti,"2",c("A","C","G","T")),])
        rlogp <- log10((errs+1)/tot)
        rlogp[is.infinite(rlogp)] <- NA
        df    <- data.frame(q=qq, errs=errs, tot=tot, rlogp=rlogp)
        mod.lo <- loess(rlogp ~ q, df, weights=log10(tot), span=2)
        pred  <- predict(mod.lo, qq)
        maxrli <- max(which(!is.na(pred))); minrli <- min(which(!is.na(pred)))
        pred[seq_along(pred)>maxrli] <- pred[[maxrli]]
        pred[seq_along(pred)<minrli] <- pred[[minrli]]
        est <- rbind(est, 10^pred)
      }
    }
  }
  MAX_ERROR_RATE <- 0.25; MIN_ERROR_RATE <- 1e-7
  est[est>MAX_ERROR_RATE] <- MAX_ERROR_RATE; est[est<MIN_ERROR_RATE] <- MIN_ERROR_RATE
  estorig <- est
  est <- est %>% data.frame() %>%
    mutate(across(everything(), ~ if_else(. < X40, X40, .))) %>% as.matrix()
  rownames(est) <- rownames(estorig); colnames(est) <- colnames(estorig)
  err <- rbind(1-colSums(est[1:3,]), est[1:3,], est[4,], 1-colSums(est[4:6,]),
               est[5:6,], est[7:8,], 1-colSums(est[7:9,]), est[9,],
               est[10:12,], 1-colSums(est[10:12,]))
  rownames(err) <- paste0(rep(c("A","C","G","T"),each=4),"2",c("A","C","G","T"))
  colnames(err) <- colnames(trans)
  return(err)
}

loessErrfun_mod2 <- function(trans) {
  qq  <- as.numeric(colnames(trans))
  est <- matrix(0, nrow = 0, ncol = length(qq))
  for (nti in c("A","C","G","T")) {
    for (ntj in c("A","C","G","T")) {
      if (nti != ntj) {
        errs  <- trans[paste0(nti,"2",ntj),]
        tot   <- colSums(trans[paste0(nti,"2",c("A","C","G","T")),])
        rlogp <- log10((errs+1)/tot)
        rlogp[is.infinite(rlogp)] <- NA
        df    <- data.frame(q=qq, errs=errs, tot=tot, rlogp=rlogp)
        mod.lo <- loess(rlogp ~ q, df, weights=tot)
        pred  <- predict(mod.lo, qq)
        maxrli <- max(which(!is.na(pred))); minrli <- min(which(!is.na(pred)))
        pred[seq_along(pred)>maxrli] <- pred[[maxrli]]
        pred[seq_along(pred)<minrli] <- pred[[minrli]]
        est <- rbind(est, 10^pred)
      }
    }
  }
  MAX_ERROR_RATE <- 0.25; MIN_ERROR_RATE <- 1e-7
  est[est>MAX_ERROR_RATE] <- MAX_ERROR_RATE; est[est<MIN_ERROR_RATE] <- MIN_ERROR_RATE
```

4.5 Learn error rates from the data

`learnErrors()` processes a subset of reads (here, up to 10^10 bases, effectively all of them) to estimate per-cycle, per-substitution error rates. This is the most computationally demanding step of the pipeline.

The results are cached to disk so that re-rendering the document does not require re-running this step.

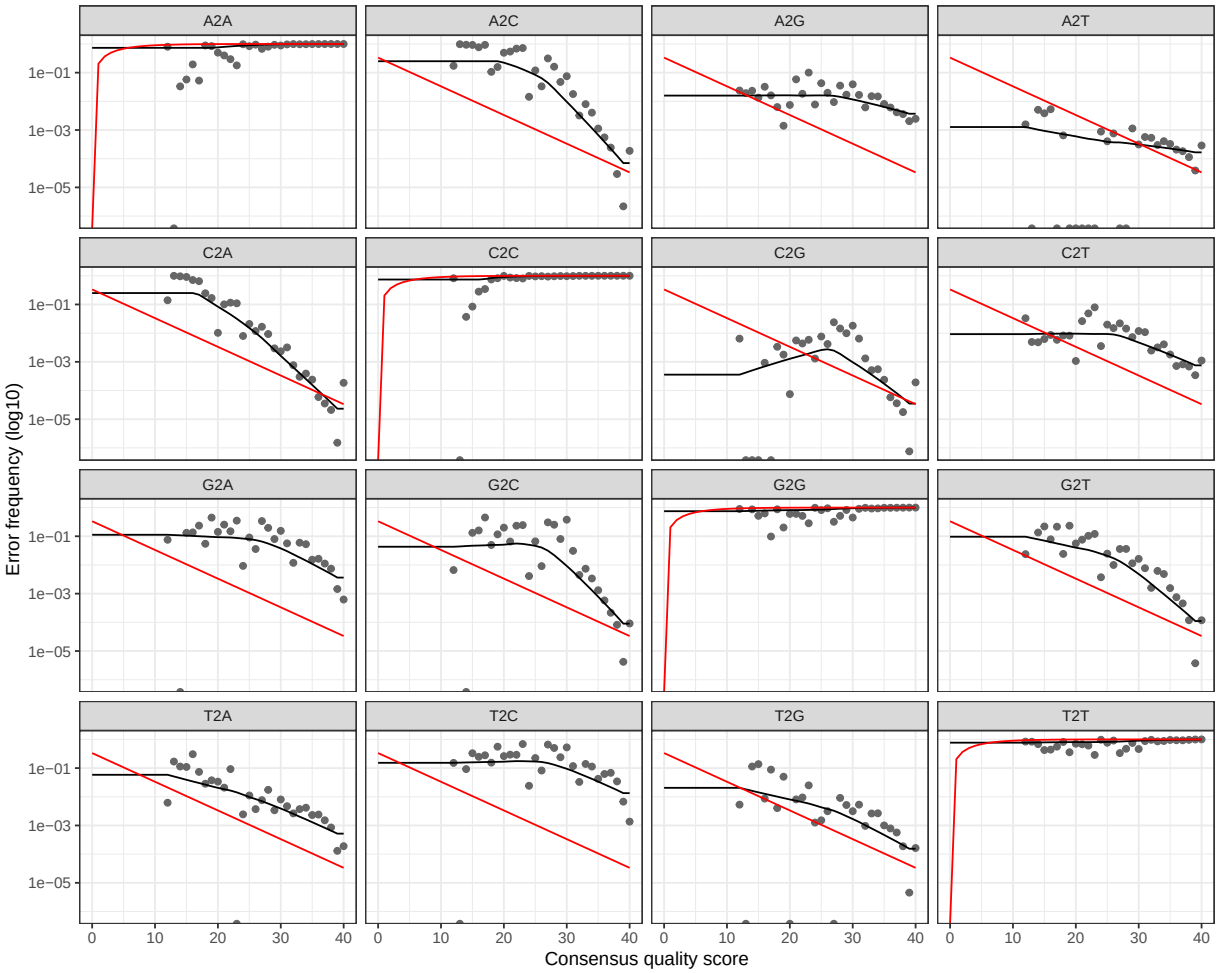
```
errF_file <- file.path(path_precomp, "errF.rds")
errR_file <- file.path(path_precomp, "errR.rds")

if (!file.exists(errF_file)) {
  message("Learning forward error model (this may take several minutes)...")
  errF <- learnErrors(filtFs, multithread = TRUE, nbases = 1e10,
                      errorEstimationFunction = loessErrfun_mod4, verbose = TRUE)
  saveRDS(errF, errF_file)
} else {
  message("Loading pre-computed forward error model.")
  errF <- readRDS(errF_file)
}

if (!file.exists(errR_file)) {
  message("Learning reverse error model (this may take several minutes)...")
  errR <- learnErrors(filtRs, multithread = TRUE, nbases = 1e10,
                      errorEstimationFunction = loessErrfun_mod4, verbose = TRUE)
  saveRDS(errR, errR_file)
} else {
  message("Loading pre-computed reverse error model.")
  errR <- readRDS(errR_file)
}
```

4.6 Visualise the error model

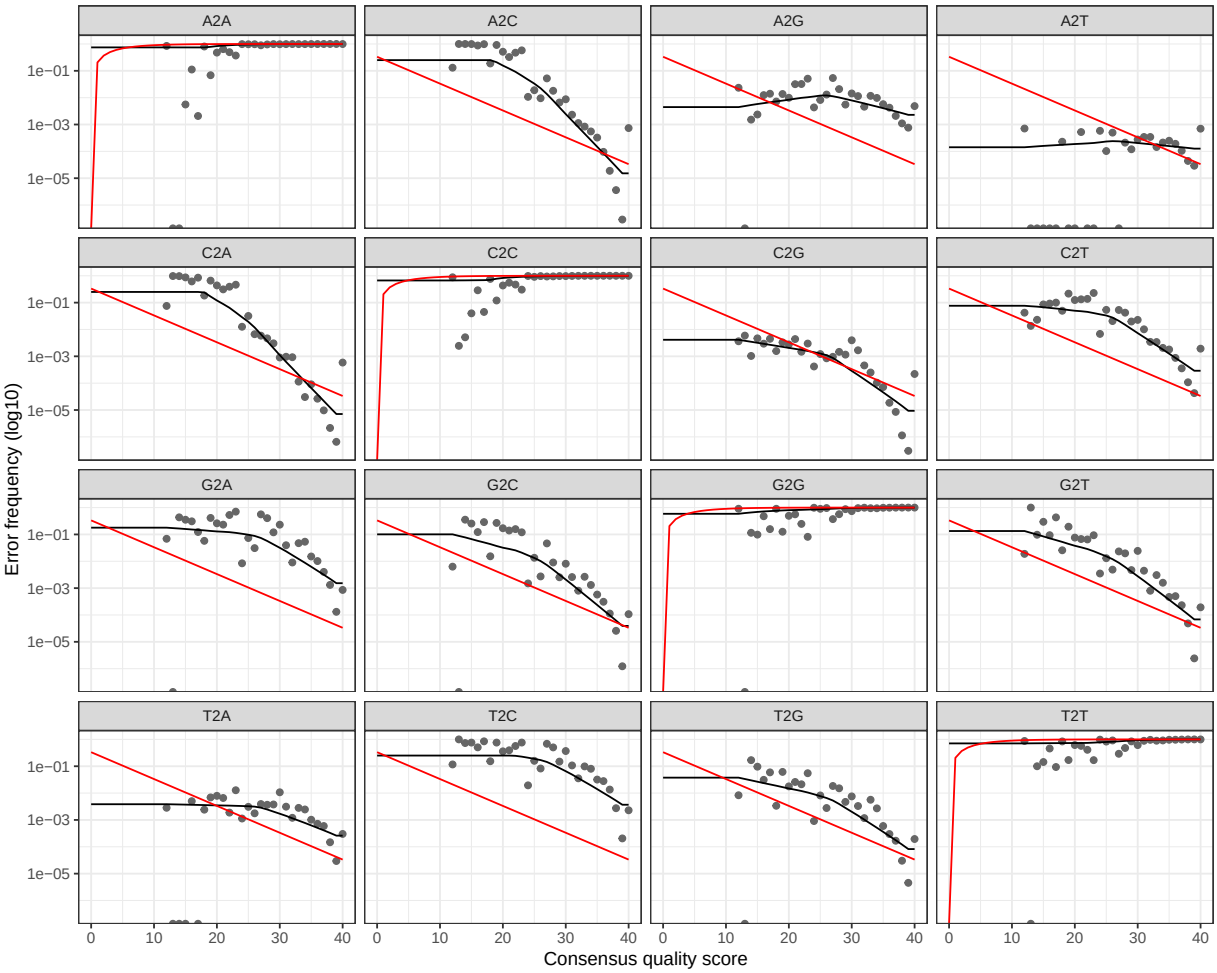
A good error model should show:

- **Black dots** (observed error rates) closely following the **black line** (fitted model)
- **Red line** (expected error rates based on the nominal quality score) close to the fitted line
- Error rates that **decrease monotonically** as quality increases

```
plotErrors(errF, nominalQ = TRUE)
```




```
plotErrors(errR, nominalQ = TRUE)
```



What a bad error model looks like

If the fitted line (black) diverges strongly from the observed points (dots), or if the curve is not monotone decreasing, the error model is unreliable. In that case, try a different option (1–3). Binned quality data often show clusters of points at a few quality levels with large gaps in between — Option 4 handles this best.

5 Denoising, Merging and Chimera Removal

5.1 Denoising with DADA2

The core DADA2 algorithm uses the error model learned in the previous chapter to distinguish true biological sequences from sequences that arose through PCR or sequencing errors. It works by asking: *given the error model, is this slightly different sequence more likely to be a genuine variant, or an error-derived copy of a more abundant sequence?*

The result is a set of **Amplicon Sequence Variants (ASVs)**: exact sequences with no artificial clustering.

```
dadaFs_file <- file.path(path_precomp, "dadaFs.rds")
dadaRs_file <- file.path(path_precomp, "dadaRs.rds")

if (!file.exists(dadaFs_file)) {
  message("Denoising forward reads...")
  dadaFs <- dada(filtFs, err = errF, multithread = TRUE)
  saveRDS(dadaFs, dadaFs_file)
} else {
  dadaFs <- readRDS(dadaFs_file)
}

if (!file.exists(dadaRs_file)) {
  message("Denoising reverse reads...")
  dadaRs <- dada(filtRs, err = errR, multithread = TRUE)
  saveRDS(dadaRs, dadaRs_file)
} else {
  dadaRs <- readRDS(dadaRs_file)
}
```

5.1.1 What does the dada2 output tell us?

```
# Inspect the first sample's denoising result
dadaFs[[1]]
```

```
dada-class: object describing DADA2 denoising results
90 sequence variants were inferred from 4843 input unique sequences.
Key parameters: OMEGA_A = 1e-40, OMEGA_C = 1e-40, BAND_SIZE = 16
```

The output reports how many unique sequences were identified in the sample and how many reads support each one. The ratio of reads to unique sequences gives an indication of how much error correction was performed.

5.2 Merging paired-end reads

After denoising, forward and reverse reads are merged to reconstruct the full amplicon sequence. Merging requires a minimum overlap region between the denoised forward and reverse reads — by default, at least 12 bp.

```
mergers_file <- file.path(path_precomp, "mergers.rds")

if (!file.exists(mergers_file)) {
  message("Merging paired reads...")
  mergers <- mergePairs(dadaFs, filtFs, dadaRs, filtRs, verbose = TRUE)
  saveRDS(mergers, mergers_file)
} else {
  mergers <- readRDS(mergers_file)
}
```

Merging failure

If a large proportion of reads fail to merge (visible in the tracking table later), the most common causes are:

1. **Insufficient overlap** — increase `truncLen` for one or both reads, or reduce it if the reads are too short
2. **Primer residues** — verify that primers were fully removed before this analysis
3. **Wrong amplicon size** — the reads must cover the amplicon with some overlap

5.3 Build the sequence table

`makeSequenceTable()` produces a sample × ASV matrix of read counts. This is the central data object in DADA2.

```
seqtab <- makeSequenceTable(mergers)
cat("Dimensions (samples × ASVs):", dim(seqtab), "\n")
```

```
Dimensions (samples × ASVs): 29 18934
```

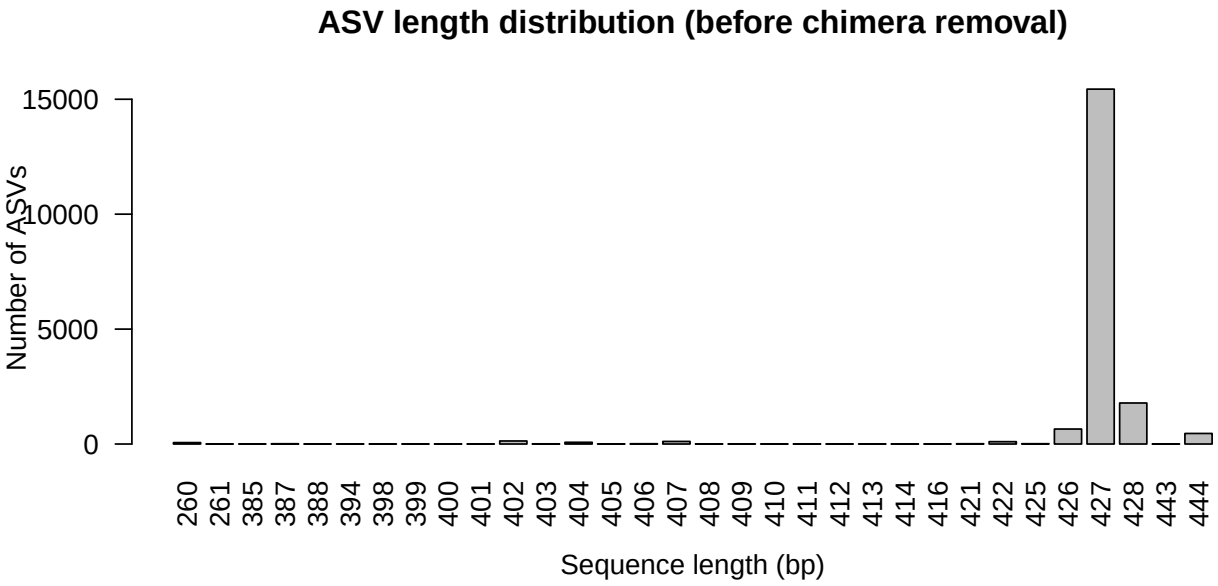
5.3.1 Sequence length distribution

The expected amplicon length for the V3–V4 region is approximately 400–470 bp. Any sequences outside this range are likely artefacts (non-specific amplification or chimeras) and will often be removed in subsequent steps.

```
seq_lengths <- table(nchar(getSequences(seqtab)))
print(seq_lengths)
```

```
  260   261   385   387   388   394   398   399   400   401   402   403   404
   62     2     1    12     1     1     1     1     2     5   132     8    75
  405   406   407   408   409   410   411   412   413   414   416   421   422
    5    12   112     7     7     5     6     3     4     3     2    12   102
  425   426   427   428   443   444
   13   652 15440  1785     2   459
```

```
# Visualise
barplot(seq_lengths,
        xlab = "Sequence length (bp)",
        ylab = "Number of ASVs",
        main = "ASV length distribution (before chimera removal)",
        las  = 2)
```



5.4 Chimera removal

Chimeric sequences are PCR artefacts created when an incomplete extension product from one cycle acts as a primer in a subsequent cycle, fusing the 5′ end of one template with the 3′ end of another. DADA2 identifies chimeras using the `removeBimeraDenovo()` function, which checks whether each sequence can be reconstructed from two more-abundant "parent" sequences.

```
seqtab_file <- file.path(path_precomp, "seqtab_nochim.rds")

if (!file.exists(seqtab_file)) {
  message("Removing chimeras...")
  seqtab.nochim <- removeBimeraDenovo(seqtab,
                                      method      = "consensus",
                                      multithread = TRUE,
                                      verbose     = TRUE)
  saveRDS(seqtab.nochim, seqtab_file)
} else {
  seqtab.nochim <- readRDS(seqtab_file)
```

```
}

cat("ASVs before chimera removal:", ncol(seqtab), "\n")
```

```
ASVs before chimera removal: 18934
```

```
cat("ASVs after  chimera removal:", ncol(seqtab.nochim), "\n")
```

```
ASVs after  chimera removal: 1091
```

```
pct_retained <- round(100 * sum(seqtab.nochim) / sum(seqtab), 1)
cat("Proportion of reads retained after chimera removal:", pct_retained, "%\n")
```

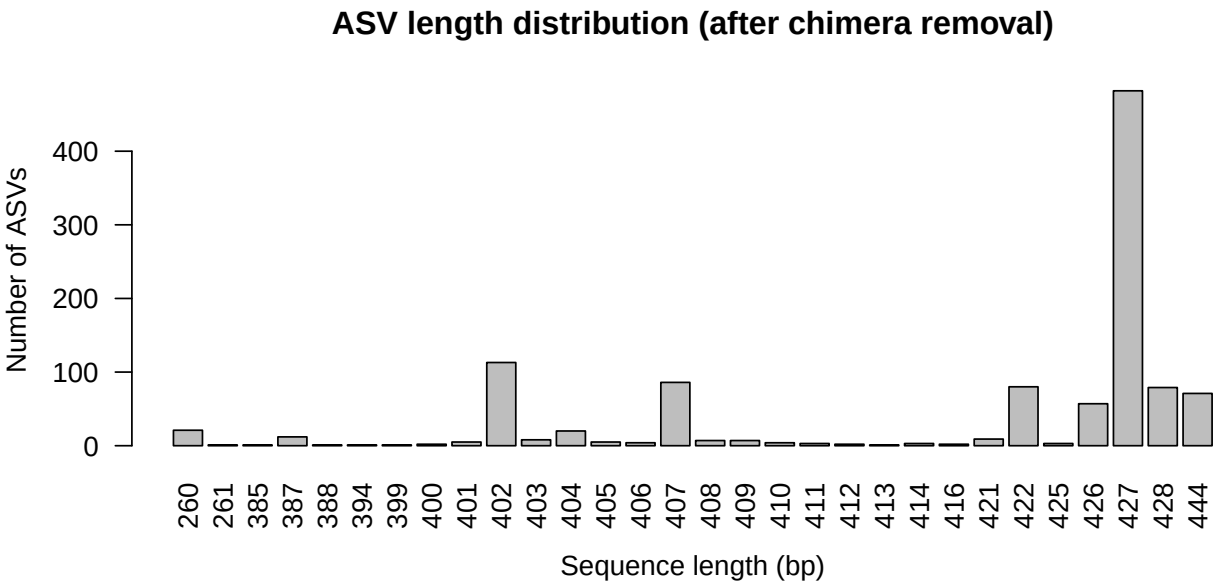
```
Proportion of reads retained after chimera removal: 89.8 %
```

> **Interpreting chimera removal**
>
> It is **normal** for a large *number* of unique sequences to be flagged as chimeric — often 10–30 % of ASVs. However, chimeric reads usually represent a small proportion of total *reads* (< 10 %), because chimeras are typically rare relative to their parent sequences. If you lose > 20 % of reads here, double-check that primers were fully trimmed.

```
seq_lengths_nc <- table(nchar(getSequences(seqtab.nochim)))
print(seq_lengths_nc)
```

```
260 261 385 387 388 394 399 400 401 402 403 404 405 406 407 408 409 410 411 412
 21   1   1  12   1   1   1   2   5 113   8  20   5   4  86   7   7   4   3   2
413 414 416 421 422 425 426 427 428 444
  1   3   2   9  80   3  57 482  79  71
```

```
barplot(seq_lengths_nc,
        xlab = "Sequence length (bp)",
        ylab = "Number of ASVs",
        main = "ASV length distribution (after chimera removal)",
        las  = 2)
```



5.5 Tracking reads through the pipeline

A key quality-control step is to track how many reads survive each stage of the pipeline. Large losses at any single step point to a specific problem.

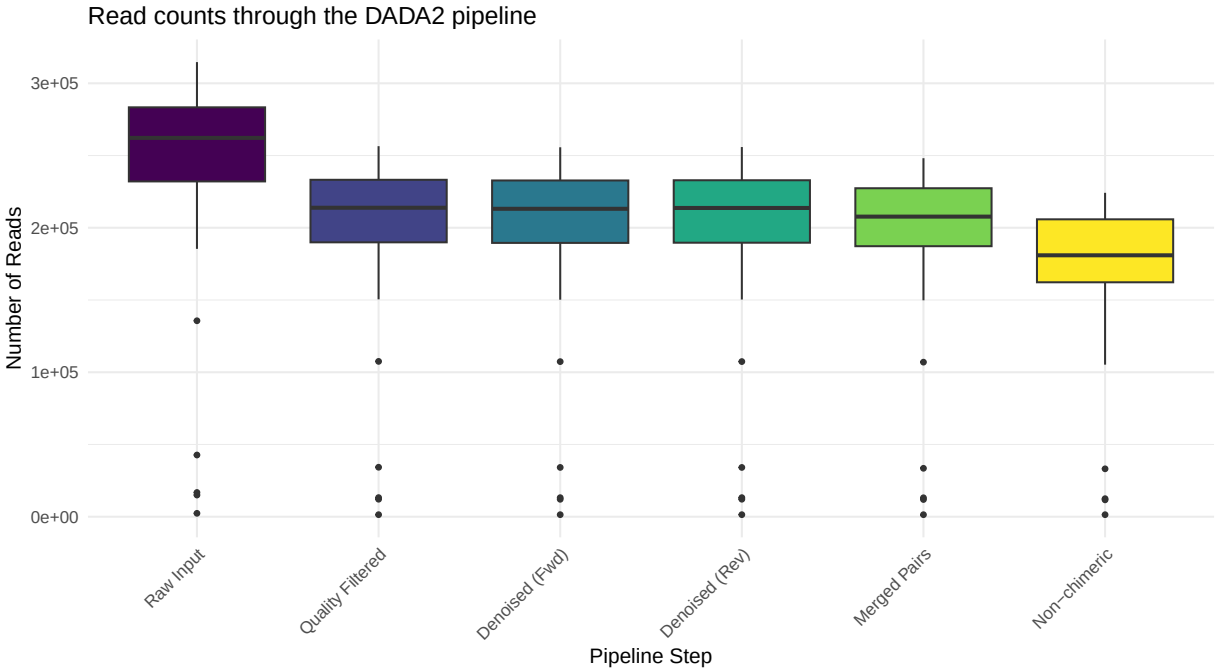
```
getN <- function(x) sum(getUniques(x))

track <- cbind(
  out,
  denoisedF = sapply(dadaFs, getN),
  denoisedR = sapply(dadaRs, getN),
  merged    = sapply(mergers, getN),
  nonchim   = rowSums(seqtab.nochim)
)
colnames(track) <- c("input", "filtered", "denoisedF", "denoisedR",
                     "merged", "nonchim")
rownames(track) <- sample.names

write.csv(track, "dada2_QC.csv")
track
```

```
                     input filtered denoisedF denoisedR merged nonchim
Commercial_R1        16778    13226     13183     13200  13113   12499
Commercial_R2       218777   178641    177733    178459 176766  176479
Commercial_R3        14993    12109     12049     12098  11922   11670
Control_extraction    2345     1419      1411      1414   1403    1403
Control_fermentation 42709    34239     34114     34105  33515   33162
Lactobacillus_H24_R1 236996   193289    192552    192803 190009  178597
Lactobacillus_H24_R2 271744   221122    220906    220980 220054  217351
Lactobacillus_H24_R3 185402   150467    150220    150328 149791  148672
Lactobacillus_H48_R1 232087   189922    189536    189665 187248  177255
Lactobacillus_H48_R2 277388   227640    227197    227476 226648  222104
Lactobacillus_H48_R3 257945   210626    210217    210457 209083  205846
```

```
Lactobacillus_W1_R1  237242   191040    190633    190702 189362  185573
Lactobacillus_W1_R2  283367   233210    232746    232883 230755  224196
Lactobacillus_W1_R3  135637   107509    107352    107386 106982  105252
Lactobacillus_W2_R1  270194   219755    219374    219573 218558  217417
Lactobacillus_W2_R2  262219   213873    213132    213708 212577  209841
Lactobacillus_W2_R3  280448   230240    229271    229942 227359  218116
Spontaneous_H24_R1   260410   209712    208384    208726 197139  162260
Spontaneous_H24_R2   297828   241630    240341    240755 230290  169513
Spontaneous_H24_R3   258467   209072    208224    208561 202745  166417
Spontaneous_H48_R1   311391   253245    252497    252931 229750  203601
Spontaneous_H48_R2   302367   243119    241992    242408 232963  186846
Spontaneous_H48_R3   295072   238579    237584    238146 230008  193478
Spontaneous_W1_R1    266958   217397    216521    216709 206641  165163
Spontaneous_W1_R2    271112   218668    218010    217981 207723  180948
Spontaneous_W1_R3    295075   239236    238162    238617 230440  187702
Spontaneous_W2_R1    291669   237723    236868    237129 225123  190377
Spontaneous_W2_R2    259494   208982    208097    208521 200762  151955
Spontaneous_W2_R3    314666   256480    255722    255872 248160  210209
```

5.5.1 Visualise read tracking

```
step_labels <- c(
  input     = "Raw Input",
  filtered  = "Quality Filtered",
  denoisedF = "Denoised (Fwd)",
  denoisedR = "Denoised (Rev)",
  merged    = "Merged Pairs",
  nonchim   = "Non-chimeric"
)

track_long <- as.data.frame(track) %>%
  rownames_to_column("Sample") %>%
  pivot_longer(-Sample, names_to = "Step", values_to = "Reads") %>%
  mutate(Step = factor(Step, levels = names(step_labels)))

ggplot(track_long, aes(x = Step, y = Reads, fill = Step)) +
  geom_boxplot(outlier.size = 1) +
  scale_x_discrete(labels = step_labels) +
  scale_fill_viridis_d() +
  labs(x = "Pipeline Step", y = "Number of Reads",
       title = "Read counts through the DADA2 pipeline") +
  theme_minimal() +
  theme(axis.text.x  = element_text(angle = 45, hjust = 1),
        legend.position = "none")
```



What to look for:

- A gradual, smooth decline across steps is expected and healthy
- A sudden large drop at the **filtering** step → truncation lengths may be too aggressive
- A large drop at **merging** → check overlap between reads; check primers are removed
- A large drop at **chimera removal** → investigate whether primers are truly absent

6 Taxonomy Assignment

6.1 Overview

Once we have a table of denoised, chimera-free ASV sequences, we can assign each one a taxonomic identity by comparing it to a reference database. DADA2 uses a **k-mer–based naive Bayesian classifier** (Wang et al. 2007) implemented in `assignTaxonomy()`.

The function assigns taxonomy from Kingdom down to Genus, returning a confidence **bootstrap value** for each level. Assignments with low bootstrap support (< 50 % by default) are reported as `NA` rather than an unreliable label.

6.2 The SILVA reference database

We use the **SILVA** database (v138), one of the most comprehensive curated 16S/18S rRNA gene databases. We use the formatted training set provided by the DADA2 team:

File	Content
`silva_nr_v138_train_set.fa.gz`	Kingdom → Genus classifier

> **Species-level assignment**
>
> Species-level assignment requires a second, separate reference file (`silva_v138_species_assignment.fa.gz`). It uses exact matching of the full ASV sequence rather than the Bayesian classifier, and should only be trusted when the amplicon region is discriminating at the species level. We skip `addSpecies()` in this tutorial because the file is not included in the dataset, but it can be added as a straightforward extra step if needed.

6.3 Run taxonomy assignment

```
taxa_file <- file.path(path_precomp, "taxa.rds")

if (!file.exists(taxa_file)) {
  message("Assigning taxonomy (this may take several minutes)...")
  taxa <- assignTaxonomy(seqtab.nochim,
                         path_silva,
                         multithread = TRUE,
                         verbose     = TRUE)
  saveRDS(taxa, taxa_file)
```

```
} else {
  message("Loading pre-computed taxonomy.")
  taxa <- readRDS(taxa_file)
}
```

6.4 Inspect the taxonomy table

```
# Remove sequence row names for readability
taxa.print <- taxa
rownames(taxa.print) <- NULL
head(taxa.print, 10)
```

```
      Kingdom    Phylum           Class                  Order
 [1,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
 [2,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
 [3,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
 [4,] "Bacteria" "Firmicutes"     "Bacilli"              "Staphylococcales"
 [5,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
 [6,] "Bacteria" "Proteobacteria" "Gammaproteobacteria" "Enterobacterales"
 [7,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
 [8,] "Bacteria" "Proteobacteria" "Gammaproteobacteria" "Enterobacterales"
 [9,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
[10,] "Bacteria" "Firmicutes"     "Bacilli"              "Lactobacillales"
      Family               Genus
 [1,] "Lactobacillaceae"   "Lactobacillus"
 [2,] "Lactobacillaceae"   "Lactobacillus"
 [3,] "Streptococcaceae"   "Lactococcus"
 [4,] "Staphylococcaceae"  "Staphylococcus"
 [5,] "Leuconostocaceae"   "Leuconostoc"
 [6,] "Enterobacteriaceae" "Enterobacter"
 [7,] "Enterococcaceae"    "Enterococcus"
 [8,] "Enterobacteriaceae" "Enterobacter"
 [9,] "Leuconostocaceae"   "Weissella"
[10,] "Lactobacillaceae"   "Pediococcus"
```

The `taxa` object is a matrix with one row per ASV and one column per taxonomic level. `NA` values indicate the classifier was not sufficiently confident at that level.

6.5 Classification summary

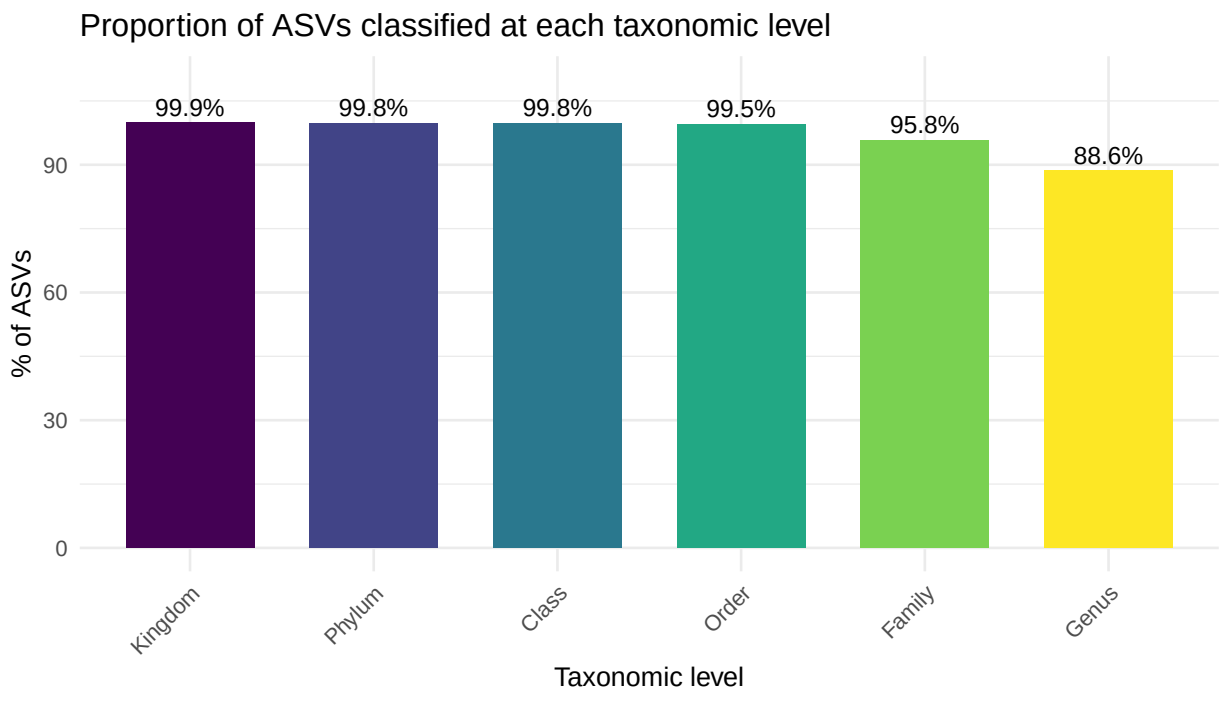
How many ASVs received a classification at each level?

```
taxa_df <- as.data.frame(taxa)
levels  <- c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus")

classif <- data.frame(
```

```
  Level      = factor(levels, levels = levels),
  ASVs_classified = sapply(levels, function(lv) sum(!is.na(taxa_df[[lv]]))),
  Total      = nrow(taxa_df)
) %>%
  mutate(Percentage = round(100 * ASVs_classified / Total, 1))

classif
```

```
          Level ASVs_classified Total Percentage
Kingdom Kingdom            1090  1091       99.9
Phylum   Phylum            1089  1091       99.8
Class     Class            1089  1091       99.8
Order     Order            1086  1091       99.5
Family   Family            1045  1091       95.8
Genus     Genus             967  1091       88.6
```

```
ggplot(classif, aes(x = Level, y = Percentage, fill = Level)) +
  geom_col(width = 0.7) +
  geom_text(aes(label = paste0(Percentage, "%")),
            vjust = -0.4, size = 3.5) +
  scale_fill_viridis_d() +
  labs(title = "Proportion of ASVs classified at each taxonomic level",
       x     = "Taxonomic level",
       y     = "% of ASVs") +
  theme_minimal() +
  theme(axis.text.x  = element_text(angle = 45, hjust = 1),
        legend.position = "none") +
  ylim(0, 110)
```



Interpreting classification depth

- Near 100 % classification at **Kingdom** and **Phylum** is expected for a clean 16S dataset
- Classification depth drops progressively at lower levels (Family → Genus) — this is normal and reflects the limits of 16S resolution
- Very low Phylum-level classification (< 80 %) may indicate: non-bacterial sequences, contamination, or a mismatch between the amplified region and the database

7 Phyloseq Integration and Export

7.1 What is phyloseq?

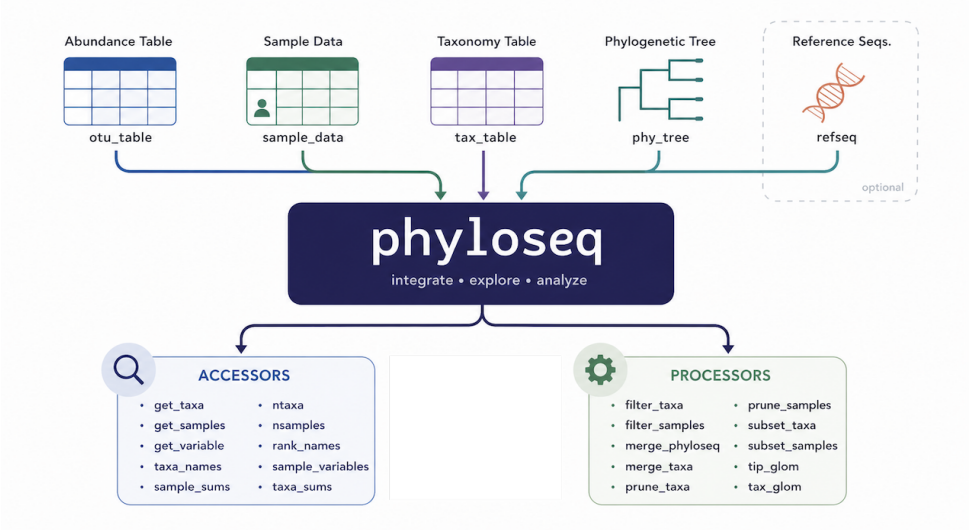


Figure 7.1: Phyloseq

phyloseq is an R package that combines three data tables into a single, consistent object:

Component	Function	Description
OTU table	`otu_table()`	ASV × sample abundance matrix
Taxonomy table	`tax_table()`	ASV × taxonomic level annotations
Sample data	`sample_data()`	Sample-level metadata
Reference sequences	`refseq()`	The actual DNA sequences (optional)

Keeping everything in one object prevents accidental misalignment between tables and gives access to a rich set of analysis and visualisation functions.

7.2 Prepare metadata

We derive metadata directly from the sample names, which encode the experimental design (`Group_Timepoint_Replicate`).

```
# Parse sample names into experimental variables
metadata <- data.frame(
  SampleID = sample.names,
  row.names = sample.names,
  stringsAsFactors = FALSE
) %>%
  mutate(
    Group = case_when(
      startsWith(SampleID, "Lactobacillus") ~ "Lactobacillus",
      startsWith(SampleID, "Spontaneous")   ~ "Spontaneous",
      startsWith(SampleID, "Commercial")    ~ "Commercial",
      startsWith(SampleID, "Control")       ~ "Control",
      TRUE ~ "Unknown"
    ),
    Timepoint = case_when(
      grepl("_H24_", SampleID) ~ "H24",
      grepl("_H48_", SampleID) ~ "H48",
      grepl("_W1_",  SampleID) ~ "W1",
      grepl("_W2_",  SampleID) ~ "W2",
      TRUE ~ NA_character_
    ),
    Replicate = as.integer(sub(".*_R(\\d+)$", "\\1", SampleID)),
    IsControl = Group == "Control"
  )

# Verify that metadata rows match the seqtab rows
stopifnot(all(rownames(seqtab.nochim) %in% rownames(metadata)))

metadata
```

	SampleID	Group	Timepoint	Replicate
Commercial_R1	Commercial_R1	Commercial	<NA>	1
Commercial_R2	Commercial_R2	Commercial	<NA>	2
Commercial_R3	Commercial_R3	Commercial	<NA>	3
Control_extraction	Control_extraction	Control	<NA>	NA
Control_fermentation	Control_fermentation	Control	<NA>	NA
Lactobacillus_H24_R1	Lactobacillus_H24_R1	Lactobacillus	H24	1
Lactobacillus_H24_R2	Lactobacillus_H24_R2	Lactobacillus	H24	2
Lactobacillus_H24_R3	Lactobacillus_H24_R3	Lactobacillus	H24	3
Lactobacillus_H48_R1	Lactobacillus_H48_R1	Lactobacillus	H48	1
Lactobacillus_H48_R2	Lactobacillus_H48_R2	Lactobacillus	H48	2
Lactobacillus_H48_R3	Lactobacillus_H48_R3	Lactobacillus	H48	3
Lactobacillus_W1_R1	Lactobacillus_W1_R1	Lactobacillus	W1	1
Lactobacillus_W1_R2	Lactobacillus_W1_R2	Lactobacillus	W1	2
Lactobacillus_W1_R3	Lactobacillus_W1_R3	Lactobacillus	W1	3
Lactobacillus_W2_R1	Lactobacillus_W2_R1	Lactobacillus	W2	1
Lactobacillus_W2_R2	Lactobacillus_W2_R2	Lactobacillus	W2	2
Lactobacillus_W2_R3	Lactobacillus_W2_R3	Lactobacillus	W2	3
Spontaneous_H24_R1	Spontaneous_H24_R1	Spontaneous	H24	1
Spontaneous_H24_R2	Spontaneous_H24_R2	Spontaneous	H24	2
Spontaneous_H24_R3	Spontaneous_H24_R3	Spontaneous	H24	3

```
Spontaneous_H48_R1     Spontaneous_H48_R1   Spontaneous       H48         1
Spontaneous_H48_R2     Spontaneous_H48_R2   Spontaneous       H48         2
Spontaneous_H48_R3     Spontaneous_H48_R3   Spontaneous       H48         3
Spontaneous_W1_R1       Spontaneous_W1_R1   Spontaneous        W1         1
Spontaneous_W1_R2       Spontaneous_W1_R2   Spontaneous        W1         2
Spontaneous_W1_R3       Spontaneous_W1_R3   Spontaneous        W1         3
Spontaneous_W2_R1       Spontaneous_W2_R1   Spontaneous        W2         1
Spontaneous_W2_R2       Spontaneous_W2_R2   Spontaneous        W2         2
Spontaneous_W2_R3       Spontaneous_W2_R3   Spontaneous        W2         3
                    IsControl
Commercial_R1           FALSE
Commercial_R2           FALSE
Commercial_R3           FALSE
Control_extraction       TRUE
Control_fermentation     TRUE
Lactobacillus_H24_R1    FALSE
Lactobacillus_H24_R2    FALSE
Lactobacillus_H24_R3    FALSE
Lactobacillus_H48_R1    FALSE
Lactobacillus_H48_R2    FALSE
Lactobacillus_H48_R3    FALSE
Lactobacillus_W1_R1     FALSE
Lactobacillus_W1_R2     FALSE
Lactobacillus_W1_R3     FALSE
Lactobacillus_W2_R1     FALSE
Lactobacillus_W2_R2     FALSE
Lactobacillus_W2_R3     FALSE
Spontaneous_H24_R1      FALSE
Spontaneous_H24_R2      FALSE
Spontaneous_H24_R3      FALSE
Spontaneous_H48_R1      FALSE
Spontaneous_H48_R2      FALSE
Spontaneous_H48_R3      FALSE
Spontaneous_W1_R1       FALSE
Spontaneous_W1_R2       FALSE
Spontaneous_W1_R3       FALSE
Spontaneous_W2_R1       FALSE
Spontaneous_W2_R2       FALSE
Spontaneous_W2_R3       FALSE
```

7.3 Build the phyloseq object

```
# Ensure tables are in the same order
metadata_matched  <- metadata[rownames(seqtab.nochim), ]
seqtab_matched    <- seqtab.nochim[rownames(metadata_matched), ]

stopifnot(all(rownames(metadata_matched) == rownames(seqtab_matched)))

otu <- otu_table(t(seqtab_matched), taxa_are_rows = TRUE)
```

```
tax <- tax_table(taxa)
sd  <- sample_data(metadata_matched)

ps <- phyloseq(otu, sd, tax)

# Store the actual DNA sequences in the phyloseq object
dna        <- Biostrings::DNAStringSet(taxa_names(ps))
names(dna) <- taxa_names(ps)
ps         <- merge_phyloseq(ps, dna)

# Rename ASVs to simple identifiers (ASV1, ASV2, ...)
taxa_names(ps) <- paste0("ASV", seq(ntaxa(ps)))

ps
```

```
phyloseq-class experiment-level object
otu_table()   OTU Table:         [ 1091 taxa and 29 samples ]
sample_data() Sample Data:       [ 29 samples by 5 sample variables ]
tax_table()   Taxonomy Table:    [ 1091 taxa by 6 taxonomic ranks ]
refseq()      DNAStringSet:      [ 1091 reference sequences ]
```

7.4 Filter unwanted taxa

16S amplification can co-amplify **mitochondrial** and **chloroplast** 16S-like sequences from plant and eukaryotic cells. These are not bacteria and must be removed before downstream analyses.

```
ps_filtered <- ps %>%
  subset_taxa(
    (is.na(Family)  | !grepl("mitochondria", Family,  ignore.case = TRUE)) &
    (is.na(Order)   | !grepl("mitochondria", Order,   ignore.case = TRUE)) &
    (is.na(Genus)   | !grepl("mitochondria", Genus,   ignore.case = TRUE)) &
    (is.na(Family)  | !grepl("chloroplast",  Family,  ignore.case = TRUE)) &
    (is.na(Order)   | !grepl("chloroplast",  Order,   ignore.case = TRUE)) &
    (is.na(Order)   | Order   != "Chloroplastida")
  )

cat("ASVs removed (mitochondria / chloroplasts):",
    ntaxa(ps) - ntaxa(ps_filtered), "\n")
```

```
ASVs removed (mitochondria / chloroplasts): 26
```

7.4.1 Remove ASVs unclassified at Family level

For most downstream community analyses we require at least Family-level classification. ASVs that could not be classified even at this level are removed.

```
ps_final <- ps_filtered %>%
  subset_taxa(!is.na(Family) & Family != "")

cat("ASVs removed (unclassified at Family):",
    ntaxa(ps_filtered) - ntaxa(ps_final), "\n")
```

```
ASVs removed (unclassified at Family): 27
```

```
cat("Final ASV count:", ntaxa(ps_final), "\n")
```

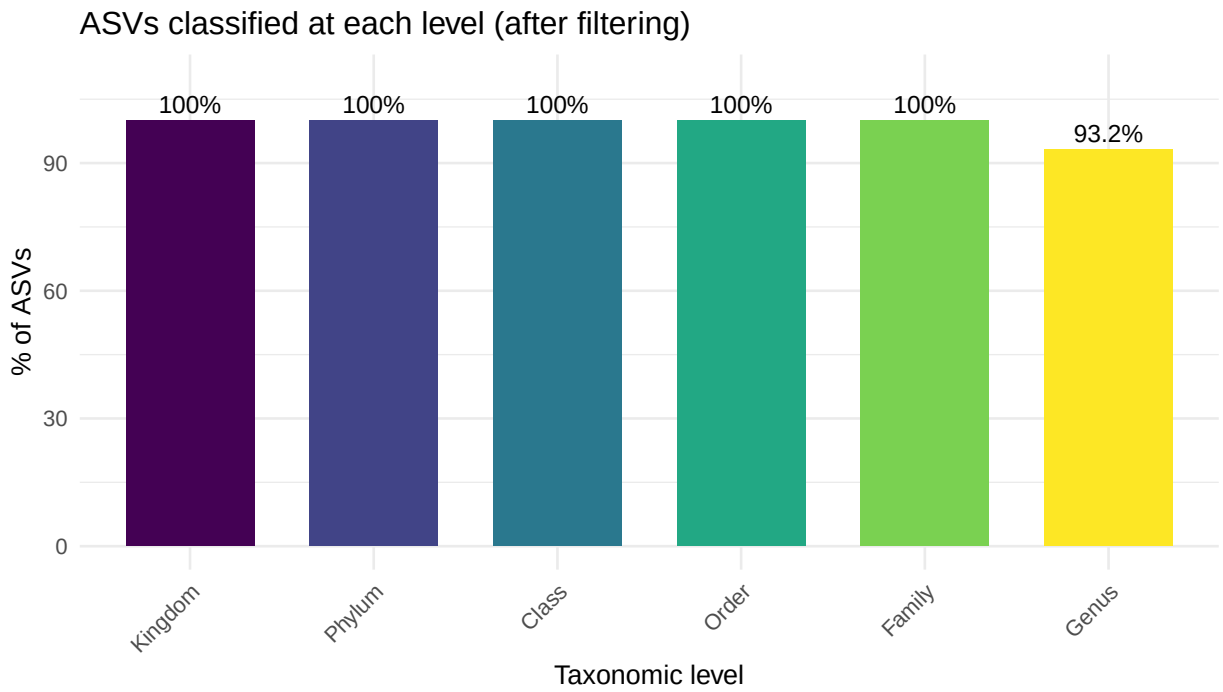
```
Final ASV count: 1038
```

```
cat("Total reads in final dataset:", sum(sample_sums(ps_final)), "\n")
```

```
Total reads in final dataset: 4674036
```

7.5 Taxonomic composition overview

```
tax_df  <- as.data.frame(tax_table(ps_final))
levels  <- c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus")
total   <- ntaxa(ps_final)

tax_classif <- data.frame(
  Level = factor(levels, levels = levels),
  Pct   = sapply(levels, function(lv) {
    round(100 * sum(!is.na(tax_df[[lv]]) & tax_df[[lv]] != "") / total, 1)
  })
)

ggplot(tax_classif, aes(x = Level, y = Pct, fill = Level)) +
  geom_col(width = 0.7) +
  geom_text(aes(label = paste0(Pct, "%")), vjust = -0.4, size = 3.5) +
  scale_fill_viridis_d() +
  labs(title = "ASVs classified at each level (after filtering)",
       x = "Taxonomic level", y = "% of ASVs") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "none") +
  ylim(0, 110)
```



7.6 Filtering summary

```
cat("\n=== FILTERING SUMMARY ===\n")
```

```
=== FILTERING SUMMARY ===
```

```
cat("After denoising and chimera removal:      ", ntaxa(ps), "ASVs\n")
```

```
After denoising and chimera removal:       1091 ASVs
```

```
cat("After removing mitochondria/chloroplasts: ", ntaxa(ps_filtered), "ASVs\n")
```

```
After removing mitochondria/chloroplasts:  1065 ASVs
```

```
cat("After removing unclassified at Family:    ", ntaxa(ps_final), "ASVs\n")
```

```
After removing unclassified at Family:     1038 ASVs
```

```
cat("Samples in final dataset:                 ", nsamples(ps_final), "\n")
```

```
Samples in final dataset:                  29
```

```
cat("Total reads in final dataset:             ", sum(sample_sums(ps_final)), "\n")
```

```
Total reads in final dataset:              4674036
```

7.7 Export outputs

```
# Phyloseq object (for downstream analysis in R)
saveRDS(ps_final, "phyloseq_object.rds")

# ASV ID → sequence mapping
asv_sequences <- data.frame(
  ASV_ID   = taxa_names(ps_final),
  Sequence = as.character(refseq(ps_final)),
  stringsAsFactors = FALSE
)
write.csv(asv_sequences, "asv_sequences_map.csv", row.names = FALSE)

# ASV taxonomy + abundance table
asv_counts   <- as.data.frame(otu_table(ps_final))
tax_table_df <- as.data.frame(tax_table(ps_final))
write.csv(cbind(tax_table_df, asv_counts),
          "asv_tax_table.csv", row.names = TRUE)

# Sample metadata + per-sample read counts per ASV
sample_abundances <- as.data.frame(t(otu_table(ps_final)))
sample_metadata   <- as.data.frame(sample_data(ps_final))
write.csv(cbind(sample_metadata, sample_abundances),
          "samples_with_abundances.csv", row.names = TRUE)

cat("Outputs saved:\n")
```

```
Outputs saved:
```

```
cat("  phyloseq_object.rds\n")
```

```
  phyloseq_object.rds
```

```
cat("  asv_sequences_map.csv\n")
```

```
  asv_sequences_map.csv
```

```
cat("  asv_tax_table.csv\n")
```

```
  asv_tax_table.csv
```

```
cat("  samples_with_abundances.csv\n")
```

```
  samples_with_abundances.csv
```

7.8 What's next?

The `phyloseq_object.rds` file is the starting point for all downstream analyses:

- **Alpha diversity** — species richness, Shannon entropy, Faith's PD
- **Beta diversity** — Bray-Curtis or UniFrac distances, ordination (PCoA, NMDS)
- **Differential abundance** — DESeq2, ANCOM-BC, MaAsLin2
- **Taxonomic bar charts** — relative abundance at Phylum or Family level
- **Network analysis** — co-occurrence networks using SpiecEasi or SPRING

These topics are covered in the downstream analysis tutorial.