# Long Read Tutorial

Jorden Rabasco

6/29/2022

In the sequencing world there are many different technologies that have advantages and disadvantages, which should all be considered before a sequencing run has begun. Short read sequencing technologies, such as illumina pair-wise sequencing, have been at the head of the field for a number of years. Illumina sequencing can give an accurate sequence but is constrained in size, with the reads being anywhere from 200-300bp. The advent of long read sequencing attempts to solve this length constraint. Pacbio one of the leaders in the long read sequencing field can generate reads much longer than traditional short read sequencing but with an increased error rate. To solve this techniques such as loop seq, and circular consensus sequencing (CCS) otherwise known as HIFI reads. These techniques sequence a genetic region many times and then creates a consensus sequence, in effect creating a large amount of coverage, in order to bring down the per base error rate. However this again creates a limitation in the size of the locus that is being sequenced. To address this and other limitations bioinformatic tools and workflows, such as those included here, have been created. This tutorial is intended for those that wish to analyze their long read sequences, generated from Pacbio techniques. The output from this workflow will consist of assigned taxonomies based on the ASVs generated by dada2. This tutorial assumes a working version of R 4.2.0

Lets begin. This tutorial utilizes data from “High-throughput amplicon sequencing of the full-length 16S rRNA gene with single-nucleotide resolution” by Callahan et al. Nucleic Acids Research, 2019c which can be found here https://academic.oup.com/nar/article/47/18/e103/5527971?login=true (https://academic.oup.com/nar/article/47/18/e103/5527971?login=true). The data itself consists of sequencing measurements of a set of nine human fecal samples from 3 different subjects, collected in a longitudinal manner, rather than having replicate measurments of the same collection. Additionally pacbio sequencing of a zymo mock community was included as a benchmark, which can be found here https://trace.ncbi.nlm.nih.gov/Traces/index.html?view=run_browser&acc=SRR9089357&display=download (https://trace.ncbi.nlm.nih.gov/Traces/index.html?view=run_browser&acc=SRR9089357&display=download). Note: once navigating to the given website you can choose whether to download the files as fastq or fasta, you will want to download them as fastq files

# Initial Set up

In the initial set up we will have to prep the reads for filtering and primer trimming. When working with the sample data you don’t need to be concerned with the following step and can move onto the next section as the reads are already preprocessed. To carry out the preprocessing steps prior to the dada2 workflow, you will need to use Pacbio software to convert multiplexed ccs reads into fastq files that can then be used in the following tutorial. Recommend settings for this preprocessing step are: –numThreads=16 –minSnr=3.75 –minReadScore=0.65 –maxLength=7000 –minLength=10 –minPasses=3 –minZScore=-5 –maxDropFraction=0.34 –minPredictedAccuracy=0.999

Prior to any R processing please make sure to install dada2 appropriately as well as the dependencies. The appropriate information can be found here: https://benjjneb.github.io/dada2/dada-installation.html (https://benjjneb.github.io/dada2/dada-installation.html)

Once dada2 is installed load the library and check the version. Make sure that you are using the latest version of dada2 as some fixes may have been implemented in later releases. This tutorial utilizes version 1.20.0 of the dada2 package.

```
library(dada2);packageVersion("dada2")
```

```
## [1] '1.20.0'
```

# Pathing setup

This section sets up the pathing for the tutorial and its output. When you first download the tutorial zip file, you will need to change the “base_path” variable to wherever the tutorial zip file is downloaded to. Note: When running your own data through this tutorial, it would be prudent to set up your data in the “data” folder of the tutorial and remove the test data present in there. This will prevent any needless errors from altered pathing to occur.

```
base_path <- "C:/Users/jorde/OneDrive/Desktop/github/Long_read_processing_tutorial" #Informs all other pathing in the tutorial
```

# Identification of Primers

In this section we need to input the primer information and initialize the rc funciton from the dada2 package. Note: After you have concluded the tutorial you will need to change “FW_pacbio” and “Rev_pacbio” to the appropriate primers used in your Pacbio sequencing.

```
Fw_pacbio<- "AGRGTTYGATYMTGGCTCAG" #change to fw primer in sequencing
Rev_pacbio <- "RGYTACCTTGTTACGACTT" #change to reverse primer in sequencing
rc <- dada2:::rc #intialized the rc function in the dada2 package
theme_set(theme_bw())
```

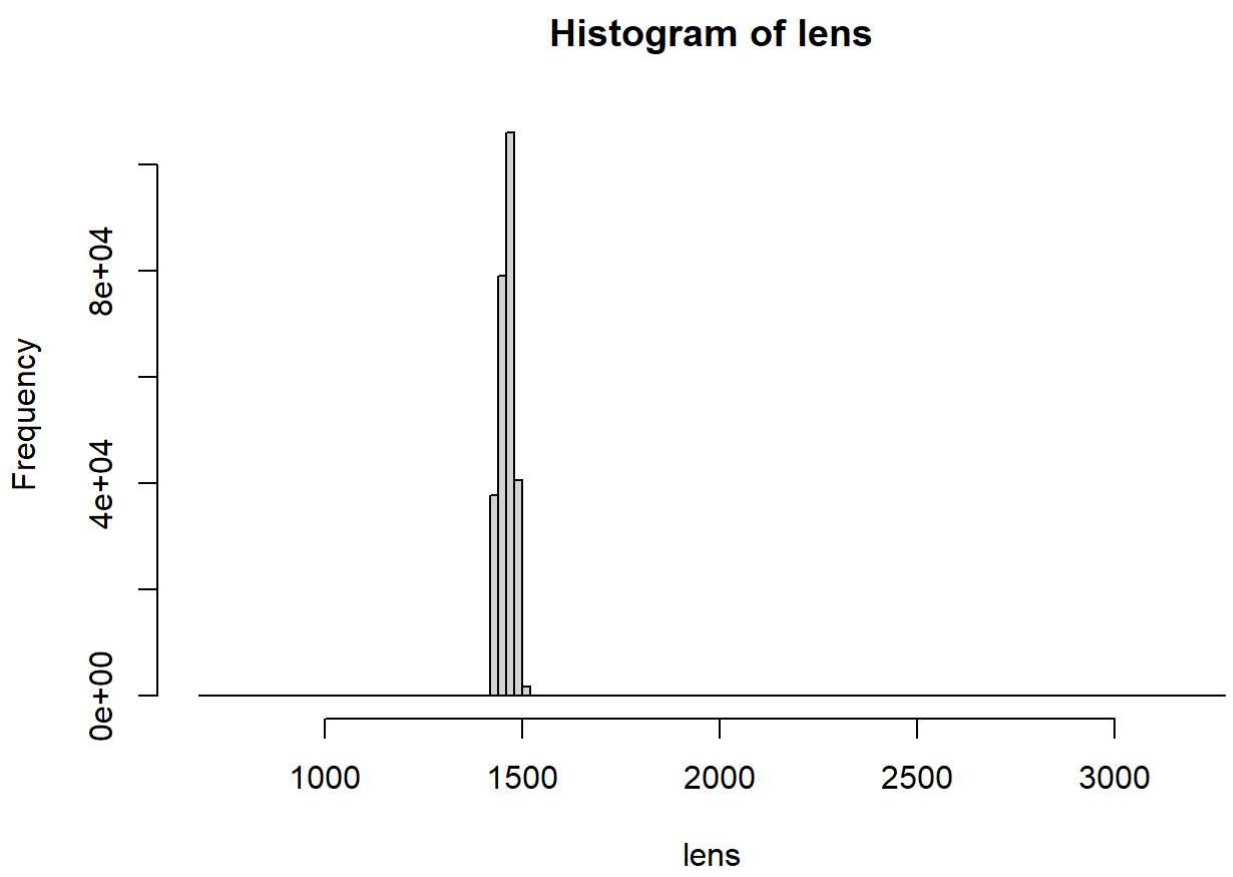
# Filtering and Primer Removal

Here we will be removing the primers and filtering out poor quality reads. We want to remove the primers as they were not part of the original sequence and thus can alter and return wrong ASVs or taxonomic assignments. Furthermore, when you have your reads in a fastq style format which is what is assumed in this workflow, each read has both an id and a sequence of quality indicators. These indicators show how sure we are that each base was identified correctly. That is to say that with low quality bases some ambiguity in the identity of the base, in that position is inherent. Therefore, we want to filter out these poor quality reads from reads that have a higher quality to ensure that the results are accurate. These two steps will ensure the quality of the reads and assist in downstream taxonomic assignments. Additionally, HiFi read orientation is mixed, meaning that some reads are in the 3’ to 5’ orientation and some are in the 5’ to 3’ orientation. This is corrected in the primer removal step, where all the reads are made to be in the same orientation. Due to this mixed orientation we will also need to reverse complement the reverse primer which can be seen in the “removePrimers” step. This is done as when the DNA amplicon is being generated the reverse primer attaches to the template DNA on the opposite strand which is in the reverse complement orientation. This means that to identify the reverse primer location we will need to look for the reverse complement of it. To ensure quality and to act as a sanity check we will want to visualize the length of the sequences after trimming to ensure that the sequences are predominantly of the lengths that we expect.

```
path<-file.path(base_path, "long_read_tutorial/data")#sets locaiton of data for input into workf
low
#The below code allows all files with the "pattern" suffix to be found in the path1 or path2 fol
der location
fn_r <- list.files(path, pattern="fastq.gz", full.names=TRUE)
nop <- file.path(base_path, "long_read_tutorial/Pacbio/noprimers", basename(fn_r)) #generates fi
le in the noprimers folder
prim <- removePrimers(fn_r, nop, primer.fwd=Fw_pacbio, primer.rev=dada2:::rc(Rev_pacbio), orient
=TRUE,  verbose=TRUE) #removes primers from fastq input file and overwrites file in noprimers fo
lder
```

```
## 9727 sequences out of 19627 are being reverse-complemented.
## 15162 sequences out of 30983 are being reverse-complemented.
## 13574 sequences out of 27505 are being reverse-complemented.
## 9274 sequences out of 18783 are being reverse-complemented.
## 15890 sequences out of 32133 are being reverse-complemented.
## 13727 sequences out of 28156 are being reverse-complemented.
## 11851 sequences out of 24760 are being reverse-complemented.
## 10566 sequences out of 21621 are being reverse-complemented.
## 7371 sequences out of 15169 are being reverse-complemented.
## 39439 sequences out of 77453 are being reverse-complemented.
```

```
lens.fn <- lapply(nop, function(fn) nchar(getSequences(fn)))
lens <- do.call(c, lens.fn)
hist(lens, 100) #generates plot of sequences after primer removal
```




There is a strong peak around 1450, so that looks good. Next we are going to filter the reads for quality.

```
filt <- file.path(base_path,"long_read_tutorial/Pacbio/filtered", basename(fn_r)) #generates fil
e in the filtered folder
track <- filterAndTrim(nop, filt, minQ=3, minLen=1000, maxLen=1600, maxN=0, rm.phix=FALSE, maxEE
=2)
track
```

```
##                     reads.in reads.out
## R11_1_P3C3.fastq.gz    17436     17436
## R3_1_P3C3.fastq.gz     26605     26605
## R3_2_P3C3.fastq.gz     24477     24081
## R3_3_P3C3.fastq.gz     16738     16738
## R9_1_P3C3.fastq.gz     28623     28623
## R9_1B_P3C3.fastq.gz    24832     24831
## R9_2_P3C3.fastq.gz     21759     21759
## R9_3_P3C3.fastq.gz     18701     18700
## R9_4_P3C3.fastq.gz     13481     13258
## Zymo_P1C1.fastq.gz     73057     72940
```

Given in the table are the reads fed into the filtering step and the output reads after the filtering step. If the numbers in each column are the same then no reads were removed due to quality, however, it is not uncommon that a few reads will be removed due to quality concerns. Note: when running your own data you will need to change parameters “minLen” and “maxLen” to encompass the length range of your expected amplicon. This will filter out any reads that have significant replication errors or vast changes in the content of the amplicon.

# Denoising

The next step is arguable the most important part of the tutorial; denoising via dada2 package. This was already loaded into the R environment in the beginning of the workflow so there is no need to worry. We will also want to produce error plots and an ASV table which we can then use for taxonomic assignment. The first step in this process is to dereplicate amplicon sequences. Then we will learn error rates, save them in an R object, and plot the output error graph. The dereplication step takes in raw fastq sequences and separates them out based on sample identification. It also generates an output R object called ‘drp’ in this case which contains a table of unique sequences, and their associated counts (‘uniques’) as well as a quality score matrix representing all quality scores that make up each unique sequence (‘qual’). The output function also contains the ‘map’ of each read to the index of its corresponding dereplicated unique sequences. To check to see if the sequences were dereplicated correctly we can look at some of the unique sequence counts in one of the samples.

```
drp <- derepFastq(filt, verbose=TRUE)
head(drp$R11_1_P3C3.fastq.gz$uniques)
```

```
## GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGAAGCGCTTGAGAAGATTTCTTCGGAATGAAACTTAAGTGACTGAGTGGCGGA
CGGGTGAGTAACGCGTGGGTAACCTGCCTCATACAGGGGGATAACAGTCAGAAATGACTGCTAATACCGCATAAGCGCACAGTACCGCATGGTACG
GTGTGAAAAACTCCGGTGGTATGGGATGGACCCGCGTCTGATTAGCTAGTTGGTGGGGTAACGGCCTACCAAGGCGACGATCAGTAGCCGGCCTGA
GAGGGCGACCGGCCACATTGGGACTGAGACACGGCCCAAACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATGCAG
CGACGCCGCGTGAGCGAAGAAGTATTTCGGTATGTAAAGCTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACTACG
TGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGACTGGCAAGTCTGATGTGAAAAC
CCGGGGGCTCAACCCCGGGACTGCATTGGAAACTGTCAGTCTAGAGTGTCGGAGGGGTAAGTGGAATTCCTAGTGTAGCGGTGAAATGCGTAGATAT
TAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGATAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGGTAG
TCCACGCCGTAAACGATGAATACTAGGTGTCGGGAAGCAAAGCTTTTCGGTGCCGCAGCAAACGCAATAAGTATTCCACCTGGGGAGTACGTTCGC
AAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTTGAC
ATCGATCTGCCCGGACTGTAACGAGTCCTTTCCCTTCGGGGACAGAGAAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGGG
TTAAGTCCCGCAACGAGCGCAACCCTTATCCTTAGTAGCCAGCAGGTAAAGCTGGGCACTCTGGGGAGACTGCCAGGGATAACCTGGAGGAAGGTG
GGGATGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCAGCCCCGCGAGGGTGAGCAAA
TCTCAAAAATAACGTCCCAGTTCGGATTGTAGTCTGCAACTCGACTACATGAAGCTGGAATCGCTAGTAATCGCAGGTCAGCATACTGCGGTGAAT
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCGGATATGCCCGAAGCCAGTGACCCAACCTTAGGGAGGGAGCTGTCGAAGGT
GGAGCCGATAACTGGGGTG
##
1320
## GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGAAGCACTTGAGAAGATTTCTTCGGAATGAAACTTAAGTGACTGAGTGGCGGA
CGGGTGAGTAACGCGTGGGTAACCTGCCTCATACAGGGGGATAACAGTCAGAAATGACTGCTAATACCGCATAAGCGCACAGTACCGCATGGTACG
GTGTGAAAAACTCCGGTGGTATGGGATGGACCCGCGTCTGATTAGCTAGTTGGTGGGGTAACGGCCTACCAAGGCGACGATCAGTAGCCGGCCTGA
GAGGGCGACCGGCCACATTGGGACTGAGACACGGCCCAAACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATGCAG
CGACGCCGCGTGAGCGAAGAAGTATTTCGGTATGTAAAGCTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACTACG
TGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGACTGGCAAGTCTGATGTGAAAAC
CCGGGGGCTCAACCCCGGGACTGCATTGGAAACTGTCAGTCTAGAGTGTCGGAGGGGTAAGTGGAATTCCTAGTGTAGCGGTGAAATGCGTAGATAT
TAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGATAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGGTAG
TCCACGCCGTAAACGATGAATACTAGGTGTCGGGAAGCAAAGCTTTTCGGTGCCGCAGCAAACGCAATAAGTATTCCACCTGGGGAGTACGTTCGC
AAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTTGAC
ATCGATCTGCCCGGACTGTAACGAGTCCTTTCCCTTCGGGGACAGAGAAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGGG
TTAAGTCCCGCAACGAGCGCAACCCTTATCCTTAGTAGCCAGCAGGTAAAGCTGGGCACTCTGGGGAGACTGCCAGGGATAACCTGGAGGAAGGTG
GGGATGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCAGCCCCGCGAGGGTGAGCAAA
TCTCAAAAATAACGTCCCAGTTCGGATTGTAGTCTGCAACTCGACTACATGAAGCTGGAATCGCTAGTAATCGCAGGTCAGCATACTGCGGTGAAT
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCGGATATGCCCGAAGCCAGTGACCCAACCTTAGGGAGGGAGCTGTCGAAGGT
GGAGCCGATAACTGGGGTG
##
879
##    GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAGCGAAGCACTTATCATTGACTCTTCGGAAGATTTGATATTTGACTGAGCGGC
GGACGGGTGAGTAACGCGTGGGTAACCTGCCTCATACAGGGGAATAACAGTTAGAAATGGCTGCTAATGCCGCATAAGCGCACAGGACCGCATGGT
CTGGTGTGAAAAACTGAGGTGGTATGAGATGGACCCGCGTCTGATTAGGTAGTTGGTGGGGTAACGGCCTACCAAGCCGACGATCAGTAGCCGGCC
TGAGAGGGTGAACGGCCACATTGGGACTGAGACACGGCCCAGACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATG
CAGCGACGCCGCGTGAAGGAAGAAGTATCTCGGTATGTAAACTTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACT
ACGTGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGAAGAGCAAGTCTGATGTGAA
AGGCTGGGGCTTAACCCCAGGACTGCATTGGAAACTGTTTTTCTAGAGTGCCGGAGAGGTAAGCGGAATTCCTAGTGTAGCGGTGAAATGCGTAGA
TATTAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGGTAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGG
TAGTCCACGCCGTAAACGATGAATACTAGGTGTCGGGGTGCAAAGCAGTTCGGTGCCGCAGCAAACGCAATAAGTATTCCACCTGGGGAGTACGTT
CGCAAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTT
GACATCTGCCTGACCGTTCCTTAACCGGAGCTTTCCTTCGGGACAGGCAAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGG
GTTAAGTCCCGCAACGAGCGCAACCCCTATCCTTAGTAGCCAGCAGTACGGCTGGGCACTCTAGGGAGACTGCCGGGGATAACCCGGAGGAAGGCG
GGGACGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCGAAGCGGTGACGCTTAGCAAA
TCTCAAAAATAACGTCCCAGTTCGGACTGCAGTCTGCAACTCGACTGCACGAAGCTGGAATCGCTAGTAATCGCGAATCAGAATGTCGCGGTGAAT
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCAGTAACGCCCGAAGTCAGTGACCCAACCTTATGGAGGGAGCTGCCGAAGGC
GGGACCGATAACTGGGGTG
```

```
##
361
##      GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGGGGAACATTTTATGGAAGCTTCGGTGGAAATAGCTTGTTCCTAGTGGC
GGACGGGTGAGTAACGCGTGGGTAACCTGCCTCACACTGGGGGATAACAGTCAGAAATGACTGCTAATACCGCATAAGCGCACGGGATTGCATGAT
CCAGTGTGAAAAACTCCGGTGGTGTGAGATGGACCCGCGTTGGATTAGCCAGTTGGCAGGGTAACGGCCTACCAAAGCGACGATCCATAGCCGGCC
TGAGAGGGTGGACGGCCACATTGGGACTGAGACACGGCCCAGACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATG
CAGCGACGCCGCGTGAAGGAAGAAGTATCTCGGTATGTAAACTTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACT
ACGTGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGAGCAGCAAGTCTGATGTGAA
AGGCAGGGGCTCAACCCCTGGACTGCATTGGAAACTGTTGATCTTGAGTACCGGAGAGGTAAGCGGAATTCCTAGTGTAGCGGTGAAATGCGTAGA
TATTAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGGTAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGG
TAGTCCACGCCGTAAACGATGAATACTAGGTGTCGGGTGGCAGAGCCATTCGGTGCCGCAGCAAACGCAGTAAGTATTCCACCTGGGGAGTACGTT
CGCAAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTT
GACATCCCGATGACCGGCACGTAACGGTGCCTTCTCTTCGGAGCATCGGAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGG
GTTAAGTCCCGCAACGAGCGCAACCCCTATCCTTAGTAGCCAGCGGTTTGGCCGGGCACTCTGAGGAGACTGCCAGGGATAACCTGGAGGAAGGCG
GGGATGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCGAGAGTGTGAGCTTAAGCAAA
TCCCAAAAATAACGTCCCAGTTCGGACTGCAGTCTGCAACTCGACTGCACGAAGCTGGAATCGCTAGTAATCGCGGATCAGAATGCCGCGGTGAAT
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCAGTAACGCCCGAAGTCAGTGACCGAACCGAAAGGACGGAGCTGCCGAAGGC
GGGACGGATGACTGGGGTG
##
357
##      GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAACGGGGAACATTTTATGGAAGCTTCGGTGGAAATAGCTTGTTCCTAGTGGC
GGACGGGTGAGTAACGCGTGGGTAACCTGCCTCACACTGGGGGATAACAGTCAGAAATGACTGCTAATACCGCATAAGCGCACGGGATTGCATGAT
CCAGTGTGAAAAACTCCGGTGGTGTGAGATGGACCCGCGTTGGATTAGCCAGTTGGCAGGGTAACGGCCTACCAAAGCGACGATCCATAGCCGGCC
TGAGAGGGTGGACGGCCACATTGGGACTGAGACACGGCCCAGACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATG
CAGCGACGCCGCGTGAAGGAAGAAGTATCTCGGTATGTAAACTTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACT
ACGTGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGAGCAGCAAGTCTGATGTGAA
AGGCGGGGGCTCAACCCCCGGACTGCATTGGAAACTGTTGATCTTGAGTACCGGAGAGGTAAGCGGAATTCCTAGTGTAGCGGTGAAATGCGTAGA
TATTAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGGTAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGG
TAGTCCACGCCGTAAACGATGAATACTAGGTGTCGGGTGGCAGAGCCATTCGGTGCCGCAGCAAACGCAGTAAGTATTCCACCTGGGGAGTACGTT
CGCAAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTT
GACATCCCGATGACCGGCACGTAACGGTGCCTTCTCTTCGGAGCATCGGAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGG
GTTAAGTCCCGCAACGAGCGCAACCCCTATCCTTAGTAGCCAGCGGTTTGGCCGGGCACTCTGAGGAGACTGCCAGGGATAACCTGGAGGAAGGCG
GGGATGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCGAGAGTGTGAGCTTAAGCAAA
TCCCAAAAATAACGTCCCAGTTCGGACTGCAGTCTGCAACTCGACTGCACGAAGCTGGAATCGCTAGTAATCGCGGATCAGAATGCCGCGGTGAAT
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCAGTAACGCCCGAAGTCAGTGACCGAACCGAAAGGACGGAGCTGCCGAAGGC
GGGACGGATGACTGGGGTG
##
339
##    GATGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAGCGAAGCACTTATCATTGACTCTTCGGAAGATTTGATATTTGACTGAGCGGC
GGACGGGTGAGTAACGCGTGGGTAACCTGCCTCATACAGGGGAATAACAGTTAGAAATGGCTGCTAATGCCGCATAAGCGCACAGGACCGCATGGT
TCGGTGTGAAAAACCGAGGTGGTATGAGATGGACCCGCGTCTGATTAGGTAGTTGGTGGGGTAACGGCCTACCAAGCCGACGATCAGTAGCCGGCC
TGAGAGGGTGAACGGCCACATTGGGACTGAGACACGGCCCAGACTCCTACGGGAGGCAGCAGTGGGGAATATTGCACAATGGGGGAAACCCTGATG
CAGCGACGCCGCGTGAAGGAAGAAGTATCTCGGTATGTAAACTTCTATCAGCAGGGAAGAAAATGACGGTACCTGACTAAGAAGCCCCGGCTAACT
ACGTGCCAGCAGCCGCGGTAATACGTAGGGGGCAAGCGTTATCCGGATTTACTGGGTGTAAAGGGAGCGTAGACGGAAGAGCAAGTCTGATGTGAA
AGGCTGGGGCTTAACCCCAGGACTGCATTGGAAACTGTTTTTCTAGAGTGCCGGAGAGGTAAGCGGAATTCCTAGTGTAGCGGTGAAATGCGTAGA
TATTAGGAGGAACACCAGTGGCGAAGGCGGCTTACTGGACGGTAACTGACGTTGAGGCTCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGG
TAGTCCACGCCGTAAACGATGAATACTAGGTGTCGGGGTGCAAAGCAGTTCGGTGCCGCAGCAAACGCAATAAGTATTCCACCTGGGGAGTACGTT
CGCAAGAATGAAACTCAAAGGAATTGACGGGGACCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCAAGTCTT
GACATCTGCCTGACCGTTCCTTAACCGGAGCTTTCCTTCGGGACAGGCAAGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGG
GTTAAGTCCCGCAACGAGCGCAACCCCTATCCTTAGTAGCCAGCAGTACGGCTGGGCACTCTAGGGAGACTGCCGGGGATAACCCGGAGGAAGGCG
GGGACGACGTCAAATCATCATGCCCCTTATGATTTGGGCTACACACGTGCTACAATGGCGTAAACAAAGGGAAGCGAAGCGGTGACGCTTAGCAAA
TCTCAAAAATAACGTCCCAGTTCGGACTGCAGTCTGCAACTCGACTGCACGAAGCTGGAATCGCTAGTAATCGCGAATCAGAATGTCGCGGTGAAT
```

```
ACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCATGGGAGTCAGTAACGCCCGAAGTCAGTGACCCAACCTTATGGAGGGAGCTGCCGAAGGC
GGGACCGATAACTGGGGTG
##
331
```

To investigate this data further we can print out the other, features of the output object by changing the code "head(drp`R11_1_P3C3.fastq.gz`uniques)" to "drp`R11_1_P3C3.fastq.gz`quals" or "drp`R11_1_P3C3.fastq.gz`map". By extension, if you wish to check the other samples you can switch the sample name in the code "head(drp `R11_1_P3C3.fastq.gz`uniques)" to whichever sample you would like to view.

To determine whether these data are appropriate for dada2 processing we will have to do some sleuthing. This is accomplished by determining how many unique sequences were identified from the total read number for a given sample. Some unique sequence need to have many copies present in order to run dada2's error model. If the data had only a few counts of each unique sequence then the data would be unsuited for dada2 processing. For example if your data produces 1000 unique sequences from 1023 total reads then it would be safe to say that the data is unsuited for this workflow and dada2 processing in general.

```
reads_out_derep<-c()
uniques_out_derep<-c()
sample_name<-c()
for (x in 1:length(rownames(track))) {
  uniques_out_derep<-c(uniques_out_derep,length(drp[[x]]$uniques))
  reads_out_derep<-c(reads_out_derep,sum(drp[[x]]$uniques))
  sample_name<-c(sample_name,rownames(track)[x])
}
derep_track<-cbind(reads_out_derep, uniques_out_derep)
rownames(derep_track)<-sample_name
derep_track
```

```
##                     reads_out_derep uniques_out_derep
## R11_1_P3C3.fastq.gz           17436               4669
## R3_1_P3C3.fastq.gz            26605               5179
## R3_2_P3C3.fastq.gz            24081               5326
## R3_3_P3C3.fastq.gz            16738               3816
## R9_1_P3C3.fastq.gz            28623               6661
## R9_1B_P3C3.fastq.gz           24831               5788
## R9_2_P3C3.fastq.gz            21759               5748
## R9_3_P3C3.fastq.gz            18700               5343
## R9_4_P3C3.fastq.gz            13258               3772
## Zymo_P1C1.fastq.gz            72940              22309
```

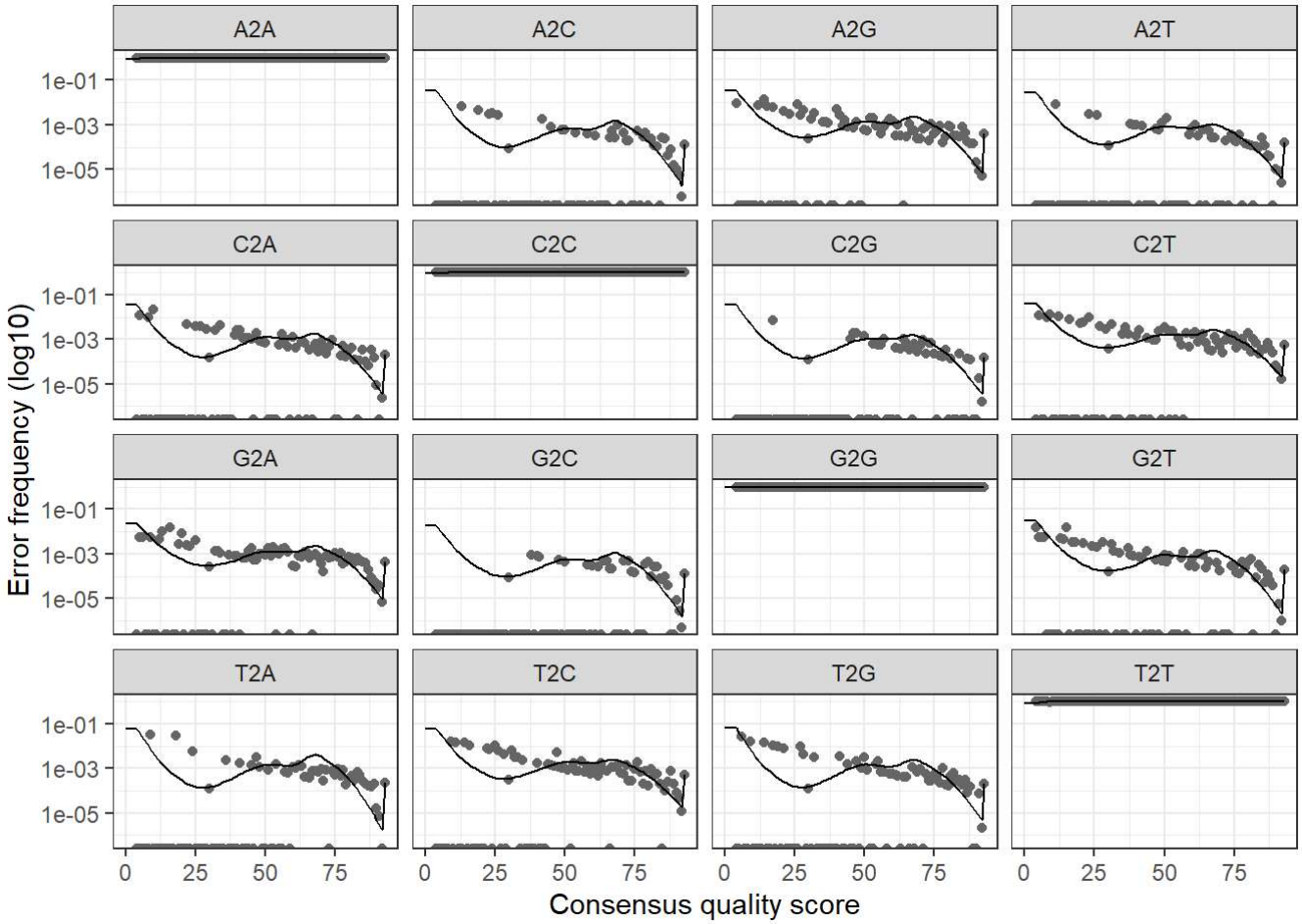
As we can see from the table above the tutorial data have many more reads for each sample than unique sequences identified, indicating that the data is well suired for the DADA2 algorithm.

The DADA2 algorithm makes use of a parametric error model (err) which generates different sets of error rates for every amplicon in the dataset. The 'learnErrors' method learns this error model from the data, by alternating the estimation of the error rates and conducting an inference of sample composition, until the sequences converge on a jointly consistent solution. As in many machine-learning problems, the algorithm must begin with an initial guess, for which the maximum possible error rates for this data are present (the error rates in the case where the most abundant sequence is correct and all the rest of the sequences are errors).

```
err <- learnErrors(drp, errorEstimationFunction=PacBioErrfun, BAND_SIZE=32, multithread=TRUE) #
 10s of seconds
```

```
## 123895827 total bases in 84860 reads from 4 samples will be used for learning the error rate
s.
```

```
saveRDS(err, file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_err.rds"))
plotErrors(err)
```




The error rates for each possible transition (A→C, A→G, …) are shown in the plot above. Points are the observed error rates for each consensus quality score. The black line shows the estimated error rates after convergence of the machine-learning algorithm. Here the estimated error rates (black line) are a good fit to the observed rates (points), and the error rates drop with increased quality as expected. Everything looks reasonable and we proceed with confidence.

Now that the error rates are learned we will finally run the star of the show dada2; utilizing both of the output R objects generated from our previous steps. There are a myriad of options included in the dada2 package, the most important of which are included in the description below but in the negative (option is not turned on). To turn on these options follow the directions included in the description of each of the options.
The first of the a fore mentioned options is the “pool” option. In this option you can decide whether or not to pool your samples when running dada2. This allows for sharing information across samples and can increase the specificity of the dada2 algorithm to resolve ASVs that maybe in low abundance. In order to turn on this option you will need to change “pool=FALSE” to “pool=TRUE”. You can also use the option “pool=psuedo”, which inputs all

ASVs detected in at least two samples in the first sample processing step as priors to the second step. Another option to mention is the “OMEGA_A” option which sets the threshold for when DADA2 calls unique sequences significantly overabundant, and therefore creates a new partition with that sequence as the center. Default is 1e-40, which is a conservative setting to avoid making false positive inferences, but which comes at the cost of reducing the ability to identify some rare variants. To change this parameter to be less conservative and more sensitive to detecting rare variants change the number in "OMEGA_A=(1*10^-40)" to a larger number. The last option that we will mention here is “DETECT_SINGLETONS”, which toggles reporting of ASVs which originated from one read (default is to only report ASVs that originate from 2 or more reads). When turned on this option will increase the sensitivity of the dada2 algorithm particularly in low abundance environments. To turn this option on change “DETECT_SINGLETONS=FALSE” to “DETECT_SINGLETONS=TRUE”. For more in-depth information regarding these options you can read the dada2 documentation found here: https://www.bioconductor.org/packages/devel/bioc/manuals/dada2/man/dada2.pdf (https://www.bioconductor.org/packages/devel/bioc/manuals/dada2/man/dada2.pdf) For examples of how these options work you can look at the sensitivity dada2 workflow found here: https://benjjneb.github.io/dada2/pseudo.html (https://benjjneb.github.io/dada2/pseudo.html)

```
dd <- dada(drp, err=err, BAND_SIZE=32, multithread=TRUE, pool=FALSE,OMEGA_A=(1*10^-40), DETECT_S
INGLETONS=FALSE) # seconds
```

```
## Sample 1 - 17436 reads in 4669 unique sequences.
## Sample 2 - 26605 reads in 5179 unique sequences.
## Sample 3 - 24081 reads in 5326 unique sequences.
## Sample 4 - 16738 reads in 3816 unique sequences.
## Sample 5 - 28623 reads in 6661 unique sequences.
## Sample 6 - 24831 reads in 5788 unique sequences.
## Sample 7 - 21759 reads in 5748 unique sequences.
## Sample 8 - 18700 reads in 5343 unique sequences.
## Sample 9 - 13258 reads in 3772 unique sequences.
## Sample 10 - 72940 reads in 22309 unique sequences.
```

```
saveRDS(dd, file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_dd.rds"))
cbind(ccs=prim[,1], primers=prim[,2], filtered=track[,2], denoised=sapply(dd, function(x) sum(x
$denoised)))
```

```
##                       ccs primers filtered denoised
## R11_1_P3C3.fastq.gz 19627   17436    17436    17242
## R3_1_P3C3.fastq.gz  30983   26605    26605    26490
## R3_2_P3C3.fastq.gz  27505   24477    24081    23846
## R3_3_P3C3.fastq.gz  18783   16738    16738    16665
## R9_1_P3C3.fastq.gz  32133   28623    28623    28396
## R9_1B_P3C3.fastq.gz 28156   24832    24831    24591
## R9_2_P3C3.fastq.gz  24760   21759    21759    21403
## R9_3_P3C3.fastq.gz  21621   18701    18700    18182
## R9_4_P3C3.fastq.gz  15169   13481    13258    12875
## Zymo_P1C1.fastq.gz  77453   73057    72940    72831
```

```
st <- makeSequenceTable(dd); dim(st)
```

```
## [1]   10 1431
```

As you can see the output table above shows the read tracking through the various steps of the workflow up until this point, such as read numbers for ccs, primer containing, filtered, and denoised reads. Additionally, if for any reason you need to stop the tutorial, the saved R object can then be loaded in and the workflow continued from this step. This can be done by running code: 'dd<-readRDS(file.path(base_path,"long_read_tutorial/Pacbio/RDS/Fecal_dd.rds"))'

# Identifying and Removing chimeras

While the core dada method corrects substitution and indel errors, chimeras still remain. Fortunately, the accuracy of sequence variants after denoising makes identifying chimeric ASVs simpler than when dealing with fuzzy OTUs. First we need to identify how many bimeras there are (if any), as well as the percentage of them in the overall reads:

```
bim <- isBimeraDenovo(st, minFoldParentOverAbundance=3.5)
# Higher MFPOA to avoid flagging intra-genomic variants
table(bim)
```

```
## bim
## FALSE  TRUE
##  1393    38
```

```
sum(st[,bim])/sum(st)
```

```
## [1] 0.004563444
```

As you can see there are 38 bimera present in the dataset, which comes out to a 0.6316% bimera rate in the reads. A relatively low bimera rate is not to be unexpected and you should only worry if the bimera rate is relatively high

# Assigning Taxonomy

Now that we have processed our data via dada2, we can plot and pull some interesting conclusions from the data. The first thing to be done is assign taxonomic assignments to the generated ASVs. This is done by utilizing the assignTaxonomy method innate to the dada2 package. This method uses trained data to inform the algorithm which taxa belong to which taxonomic assignments. To accomplish this we need a source taxonomy file which for this sample dataset can be found here: https://zenodo.org/record/4587955/files/silva_nr99_v138.1_train_set.fa.gz?download=1 (https://zenodo.org/record/4587955/files/silva_nr99_v138.1_train_set.fa.gz?download=1). You will then have to move this file into the tax folder found within the tutorial folder. Then we run the main function and voilà! We have taxonomies!

```
tax_path <- file.path(base_path, "long_read_tutorial/tax/silva_nr99_v138.1_train_set.fa.gz")
tax <- assignTaxonomy(st, tax_path, multithread=TRUE)
tax[,"Genus"] <- gsub("Escherichia-Shigella", "Escherichia", tax[,"Genus"]) # Reformat to be com
patible with other data sources
head(unname(tax))
```

```
##      [,1]       [,2]               [,3]                 [,4]
## [1,] "Bacteria" "Verrucomicrobiota" "Verrucomicrobiae"    "Verrucomicrobiales"
## [2,] "Bacteria" "Firmicutes"        "Negativicutes"       "Acidaminococcales"
## [3,] "Bacteria" "Proteobacteria"    "Gammaproteobacteria" "Enterobacterales"
## [4,] "Bacteria" "Firmicutes"        "Bacilli"             "Bacillales"
## [5,] "Bacteria" "Firmicutes"        "Bacilli"             "Staphylococcales"
## [6,] "Bacteria" "Firmicutes"        "Bacilli"             "Lactobacillales"
##      [,5]                 [,6]
## [1,] "Akkermansiaceae"    "Akkermansia"
## [2,] "Acidaminococcaceae" "Acidaminococcus"
## [3,] "Enterobacteriaceae" "Salmonella"
## [4,] "Bacillaceae"        "Bacillus"
## [5,] "Staphylococcaceae"  "Staphylococcus"
## [6,] "Listeriaceae"       "Listeria"
```

# Data Management

In this next section we will be saving our processed data as R objects. An Rdata file either an '.Rdata' or '.rds' file can be used to store R objects in a format native to R. There are multiple advantages of saving this way when contrasted with non-native storage approaches including; faster to restoration of the data to R and R specific information encoded in the data (e.g., attributes, variable types, etc) is preserved. The files are saved here in case you would like to stop the tutorial here or continue on another analysis path.For those wishing to continue on with the tutorial the data is also reloaded into the R environment.

```
#save R objects
saveRDS(st, file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_st.rds"))
saveRDS(tax, file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_tax_Silva128.rds"))

#relaod R objects
st <- readRDS(file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_st.rds"))
tax <- readRDS(file.path(base_path, "long_read_tutorial/Pacbio/RDS/Fecal_tax_Silva128.rds"))
```

# Sample metadata loading

Import the metadata for the samples to facilitate downstream processing and visualizations. The metadata file consists of a 4 column csv file with headers on the first line of the file. The column headers are as follows: X, SampleOrder, Subject, and SampleID. Make sure the the rownames for both the df object and the st object are the same to facilitate further analysis. This will be the sample name for the graphing analysis which is generated from the input file names removing the ".fastq.gz" extension.

```
pac_path_metadata<-file.path(base_path, "long_read_tutorial/fecal_metadata.csv")
ft <- sweep(st, 1, rowSums(st), "/")
df<-read.csv(file = pac_path_metadata)
rownames(df) <- df$X
head(df)
```

```
##                     X SampleOrder Subject SampleID
## R3_1_P3C3   R3_1_P3C3           1      R3    R_3.1
## R3_2_P3C3   R3_2_P3C3           2      R3    R_3.2
## R3_3_P3C3   R3_3_P3C3           3      R3    R_3.3
## R9_1_P3C3   R9_1_P3C3           1      R9    R_9.1
## R9_1B_P3C3 R9_1B_P3C3           1      R9   R_9.1B
## R9_2_P3C3   R9_2_P3C3           2      R9    R_9.2
```

```
rownames(st)<-gsub(".fastq.gz", "", rownames(st))
```

# Inspect E.coli

Okay now that we have denoisd our data it is time to draw some conclusions and visualizations from the work we have just done. To see how many E. coli strains are present per sample we need to first retrieve the sequences of the ASVs, which are stored in the ‘st’ table and then extracted via the ‘getseqeunces()’ method. Then to check to see which sequences were identified as E.coli we search for ‘Escherichia’ or ‘E.coli’ in the table generated in the previous step. After this we sum up all the identified E.coli rows and print out the table as can be seen below.

```
sq <- getSequences(st)
is.ecoli <- tax[,"Genus"] %in% "Escherichia"
sqec <- sq[is.ecoli]
which(is.ecoli)
```

```
##  [1]    7   18   22   30   34   41   46   68   71   74   75   81   83   84   85
## [16]   90  459  471  497  682 1185
```

```
rowSums(st[,is.ecoli]>0)
```

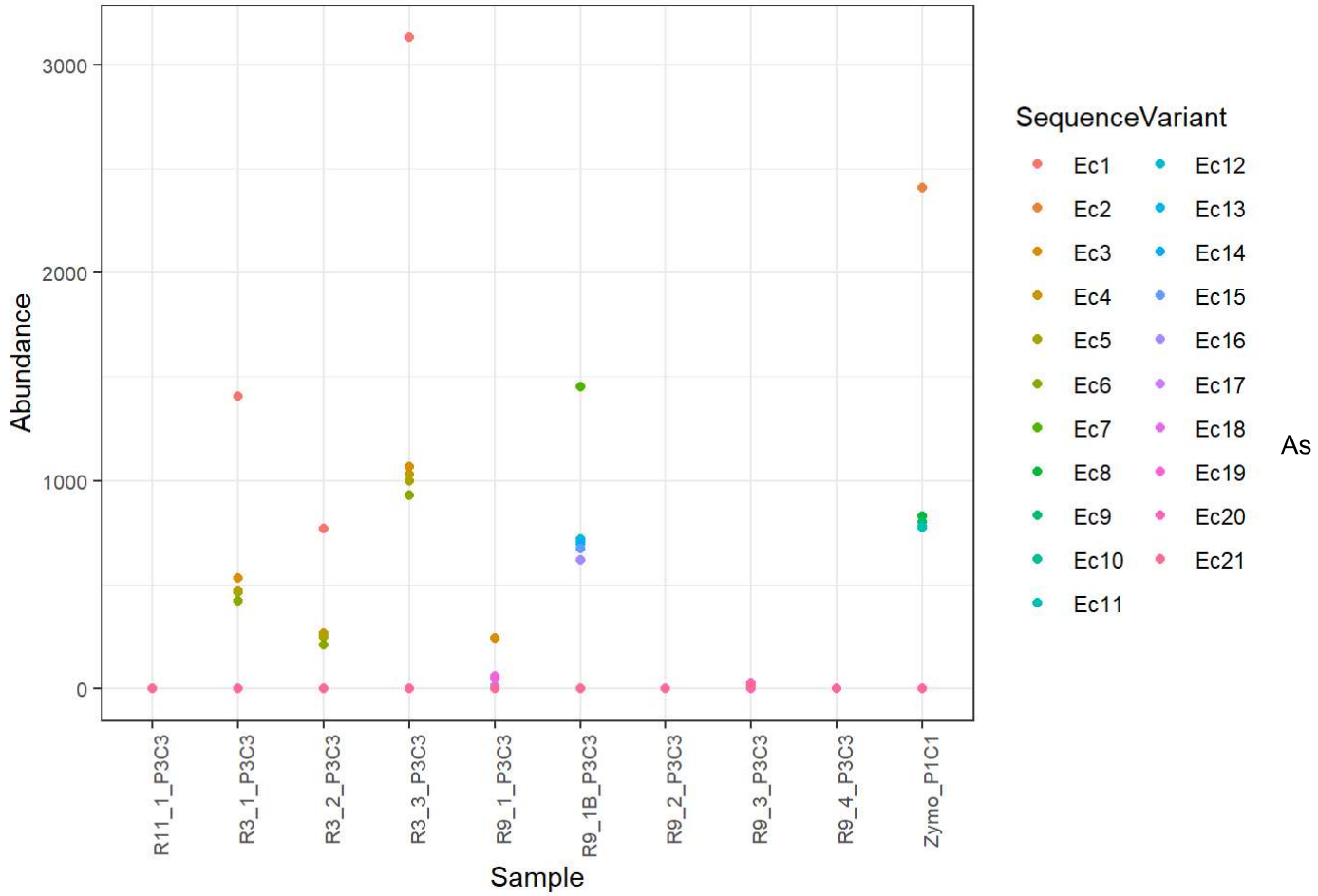
```
## R11_1_P3C3  R3_1_P3C3  R3_2_P3C3  R3_3_P3C3  R9_1_P3C3 R9_1B_P3C3  R9_2_P3C3
##          0          5          5          5         10          6          0
##  R9_3_P3C3  R9_4_P3C3  Zymo_P1C1
##          2          0          5
```

As you can see there are some samples that have several ASVs that were identified as E.coli alleles.This is unsurprising as the data we are dealing with in the tutorial is 16S longread sequencing data meaning that we would expect anywhere from 1-7 different alleles to be present in a sample if there is one E.coli strain. This is due to a standard E.coli strain having multiple (7) 16S operons present in their genome. These operons can be duplications of each other or they could be different leading to the the amount of expected alleles to be from 1-7. If you find that there were a large amount of alleles identified from one sample (>21) it is likely that there is a

contamination issue happening at the sequencing level. It is unlikely that there would be a large number of strains in a single sample as between strains the 16S operons are relatively conserved. Additionally it is worth noting that short read sequencing cannot typically resolve these different allelic differences within a strain.

Now we can visualize the distribution of the E.coli variants

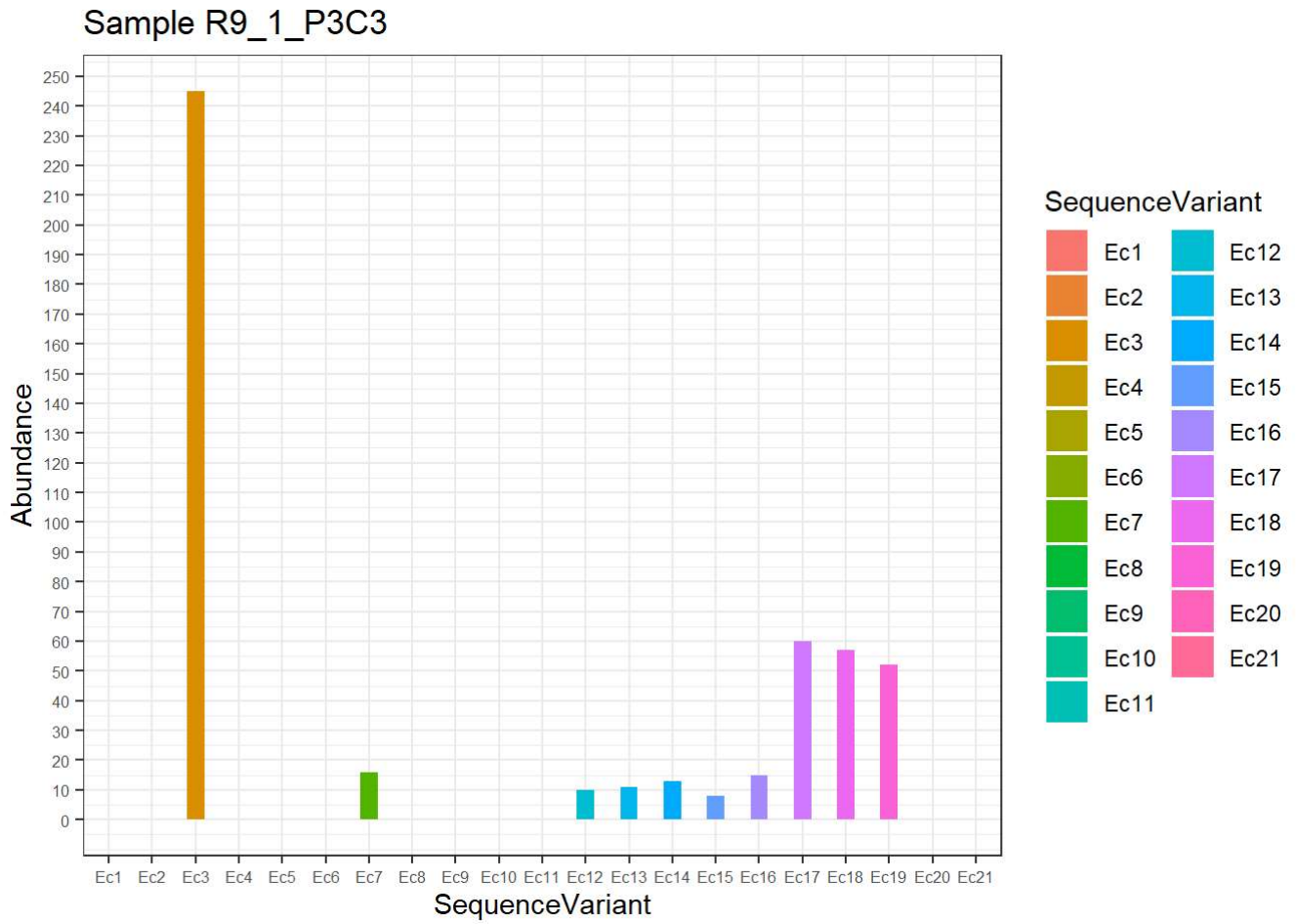
```
ecdf <- data.frame(st[,is.ecoli])
names(sqec) <- paste0("Ec", seq(ncol(ecdf)))
ecnames <- names(sqec); names(ecnames) <- sqec # map to ecnames from sequences
colnames(ecdf) <- names(sqec)
ecdf <- cbind(ecdf, Sample=rownames(st))
ecm <- melt(ecdf, id.vars="Sample", value.name="Abundance", variable.name="SequenceVariant")
plot<-ggplot(data=ecm, aes(x=Sample, y=Abundance, color=SequenceVariant)) + geom_point()
plot+ theme(axis.text = element_text(size=8), axis.text.x = element_text(angle=90, hjust=1))
```




As you can see from the graph above each Ecoli variant denoted as Ec1,Ec2,ect, are identified and denoted on the graph along with their abundance on the y-axis. When looking at the test data you can tell the subject R3 has similar strain presence across all three time point collections allbeit in differing abundances. It is also worth noting that sample R9_1 has 10 strains present but in relatively low abundances making the differentiation of the strain plotting more difficult. To remedy this we will need to take a closer look at that sample individually.

To look closer at an individual sample when not using the tutorial you will only need to change the line ‘isam <- “R9_1_P3C3”’ to whatever sample you want to see in more detail. For example if I wanted to see sample R3_1_P3C3 in more detail I would use ‘isam <- “R3_1_P3C3”’ instead of the given line of code. In the graph presented below we are showing the adundances of each strain within a sample as a bar graph instead of a stacked dot plot as seen above.

```
isam <- "R9_1_P3C3"
plot<-ggplot(data=ecm[ecm$Sample == isam,], aes(x=SequenceVariant, y=Abundance, fill=SequenceVar
iant)) +
  geom_col(width=0.4) + scale_y_continuous(breaks=seq(0,250,10)) +
  ggtitle(paste("Sample", isam) )
plot + theme(axis.text = element_text(size = 6))
```




When analyzing the graph for the tutorial data and your own data it is important to keep in mind that an Ecoli genome has 7 copies of a 16S operon some of which maybe exact duplicates. With this in mind we can assess the graph and say that since Ec3 has ~4x abundance of Ec17, Ec18, and Ec19 then they are probably sourced from the same strain where there are 4 exact copies of the Ec3 allele in the genome and one copy of the other given alleles. You can look for patterns like this in your own data to pull out relevant conclusions about the present strains.

To look further into this the exact abundance of each strain shown in the plot above can then be seen in a tabular format. To augment this to fit your own data you will need to change ‘ecdf[“R9_1_P3C3”,]’ to include the name of the sample you would like to see in a table as it appears on the graph above. This table and the above graph are displaying the relative abundances of each strain in the sample meaning that with an incomplete sampling of a microbial population these are the amount of each strain present in that sample. Thus we can infer that in the full population these relative abundances hold true (i.e. that strain Ec3 will have a relative abundance ~15x that of Ec7)

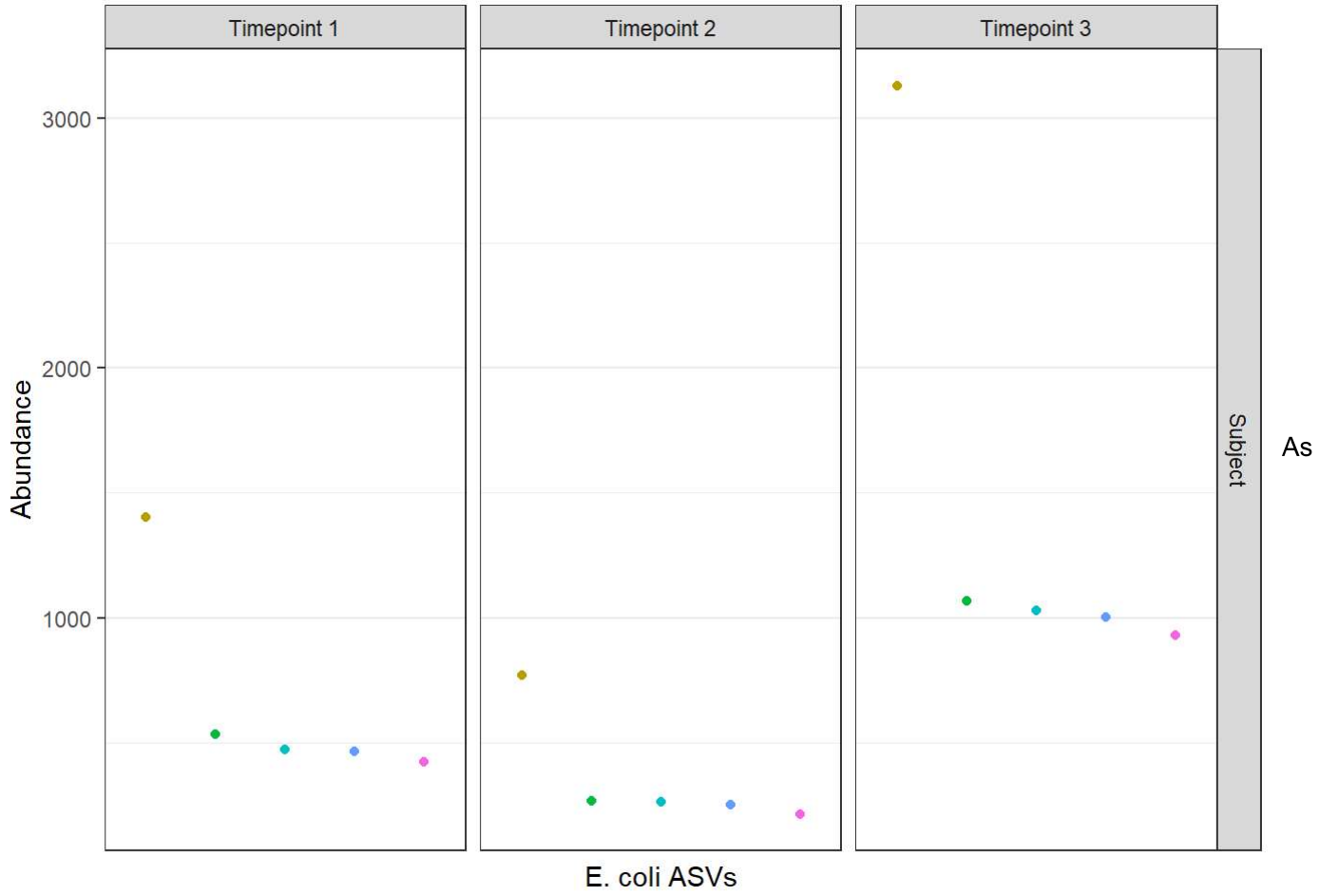
```
ecdf["R9_1_P3C3",]
```

```
##           Ec1 Ec2 Ec3 Ec4 Ec5 Ec6 Ec7 Ec8 Ec9 Ec10 Ec11 Ec12 Ec13 Ec14 Ec15
## R9_1_P3C3   0   0 245   0   0   0  16   0   0    0    0   10   11   13    8
##           Ec16 Ec17 Ec18 Ec19 Ec20 Ec21    Sample
## R9_1_P3C3   15   60   57   52    0    0 R9_1_P3C3
```

Here we show a plot of the present Ecoli ASVs of a given subject across time points. For this section make sure that your metadata file is in the same format as the given one in this tutoiral. This plot will show all of the present ASVs across all time points color coded. With the abundance on the y-axis and the differing Ecoli ASVs on the x-axis.

```
sample_order_arr<-c()
sample_sub_arr<-c()
for (sample_id in ecm$Sample) {
  sample_order_arr<-c(sample_order_arr, df[df$X == sample_id,]$SampleOrder)
  sample_sub_arr<-c(sample_sub_arr, df[df$X == sample_id,]$Subject)
}
ecm$SampleOrder<-cbind(sample_order_arr)
ecm$Subject<-cbind(sample_sub_arr)
ecm$Timepoint <- paste("Timepoint", ecm$SampleOrder)

ecmp <- ecm[ecm$Subject %in% c("R3"),] #change "R3" to whichever subject you would like plotted
 across time
ecmp$SequenceVariant <- as.character(ecmp$SequenceVariant)
ecmp <- ecmp[ecmp$Abundance > 0,]
ecmp$X<-gsub("Ec", "", ecmp$SequenceVariant)
dflim <- data.frame(X=c(1, 1), Abundance=c(3000, 1500),
                    SequenceVariant=c(""),
                    SubjectLabel=c("Subject"), Timepoint=c("Timepoint 1"))
p.ecoli <- ggplot(data=ecmp, aes(x=X, y=Abundance, color=SequenceVariant)) + geom_point() +
  facet_grid(SubjectLabel~Timepoint, scales="free_y") +
  xlab("E. coli ASVs") + theme(axis.ticks.x=element_blank(), axis.text.x = element_blank()) +
  theme(panel.grid.major.x=element_blank(), panel.grid.minor.x=element_blank()) +
  geom_blank(data=dflim) +
  guides(color=FALSE)
p.ecoli
```




As we can see the abundance magnitudes change and are available across all three time points, however the ASV abundances relative to one another are relatively stable. This is to be expected baring any drastic intervention, or event which could change the relative abundance of the microbial species in the microbiome. Raw abundance counts can be very variable from sequencing run to sequencing run depending on a variety of factors. By looking at the relative abundances instead we can get a clearer picture of the microbial community free of the artifacts on sequencing.

Other detailed plots and visualizations can be seen here https://benjjneb.github.io/LRASManuscript/LRASms_fecal.html (https://benjjneb.github.io/LRASManuscript/LRASms_fecal.html)

This now concludes our tutorial! Throughout this workflow we have taken raw sequencing data, processed it using cutting edge methods, and then visualized that data in a pleasing and nuanced manner. Now you are ready to go out into the world of long read analysis and try this workflow on your own data. Good luck out there!