Analyzing fish eDNA sequences using DADA2 pipeline

Overview

The goal is to adapt the DADA2 pipeline to Mark Stoeckle's 12S experiment. Sample sequences will be identified using 12S reference file containing sequences of 262 unique vertebrates found around New York. The starting point is a set of Illumina-sequenced paired-end fastq files that have been split (or demultiplexed) by sample and which have barcodes/adapters already removed. The end product will be a sequence table, analogous to the ubiquitous "OTU table", which records the number of times sample sequences were observed in each sample. The key difference between the output of DADA2 and standard OTU analyses is that DADA2 infers sample sequences exactly rather than clustering sequences into fuzzy OTUs which hide and complicate biological variation.

Anything following "###" are comments or instructions that should not be run as commands.
Anything following a single "#" should be run in specific instances but not each analysis.
Anything following "##" is an example of potential output for reference.
Anything in a textbox is expected output. Anything not preceded by # or inside a textbox should be run as a command.

Getting ready: Installing Software and Loading Packages

First the R and RStudio software, where this script will be run, must be downloaded.

Mac users can download R using the site:
https://cran.r-project.org/bin/macosx/
Download the most recent version (The first hyperlink under "Latest release")

Windows users can download R using the site:
https://cran.r-project.org/bin/windows/
Download the most recent version (The first hyperlink under "Latest release")

Go to https://www.rstudio.com/products/rstudio/download/#download and download the most recent RStudio software appropriate to your operating system. Once downloaded, open file and install as instructed (in Mac, drag the "RStudio" icon into "Applications".

If you plan to blast the sample files against the reference database, the NCBI Blast+ software must be installed to your hard drive. If you are not making a taxonomy table with the names of identified sequences, you can skip this installation. Late in the workflow you will create the reference database in R and blast the sample sequences against it. Installation instructions can be found on the NCBI website:
https://www.ncbi.nlm.nih.gov/books/NBK569861/

```
###You will be directed to the link:
### https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/
### where you will download the software ending in ".tar.gz" (NOT
.tar.gz.md5) appropriate for your operating system (macosx, win64, linux,
etc.) as well as the installer ending in .dmg ("NOT .dmg.md5"). Once it
is downloaded, unzip the file and move to your working directory where
you will be storing the sequence data. Open the .dmg installer and follow
the instructions to install the NCBI+ software. Make sure to check that
the software was installed correctly using your terminal. In macosx for
example you can open the Terminal (Application > Utilities > Terminal)
and type "blastn -help". It should return with numerous lines under
#"USAGE" as well as the DESCRIPTION "Nucleotide-Nucleotide BLAST
2.12.0+". If you receive an error such as "Bad CPU", make sure to
reinstall from the website, as this script will not run without the NCBI
Blast+ software.

### Download 12S reference file here. 12S is ribosomal RNA (rRNA)
responsible for translation of messenger RNAs into mitochondrial protein
and reliable vertebrate marker. This reference file was developed by Mark
Stoeckle and contains 262 unique vertebrate sequences commonly found
around New York.
### before starting this pipeline, make sure you put reference file
"12S_ECO_refs_for_DADA2.fas" and all fastq sample files in the same
folder called "data-raw"

###Every few months you should uninstall and reinstall the packages you
will be using to ensure they are most recent versions.

#remove.packages("BiocManager")

#remove.packages("dada2")

#remove.packages("ShortRead")

#remove.packages("ggplot2")

#remove.packages("phylosec")

#remove.packages("Biostrings")

#remove.packages("dplyr")

#remove.packages("easycsv")

#remove.packages("tidyverse")

#remove.packages("devtools")

#remove.packages("rBLAST")


### Installing and Loading Packages
```

```
### When it prompts you to “Update all/some/none? [a/s/n]” packages,
choose “n”

#install.packages("BiocManager", force = TRUE)

#BiocManager::install("dada2", force = TRUE)

#BiocManager::install("ShortRead", force = TRUE)

#BiocManager::install("ggplot2", force = TRUE)

#BiocManager::install("phyloseq", force = TRUE)

#BiocManager::install("Biostrings", force = TRUE)

#BiocManager::install("dplyr", type = "binary", force = TRUE)

#BiocManager::install("easycsv", force = TRUE)

#BiocManager::install("tidyverse", force = TRUE)

#install.packages("devtools")

#devtools::install_github("mhahsler/rBLAST", force = TRUE)



### Each time it prompts you to restart, click NO


### If any packages do not install properly due to errors, you can force
install packages using the following command. Replace “dada2” with
whatever package you are having trouble installing.

#install bioconductor
#  if (!require("BiocManager", quietly = TRUE))
# install.packages("BiocManager")
# BiocManager::install(version = "3.10")

#BiocManager::install("rBLAST", type = "binary")

#if (!requireNamespace("BiocManager", quietly = TRUE))
#      install.packages("BiocManager")
#      BiocManager::install("dada2", force = TRUE)

### when it says “update all/some/none?” Click none
### if it says “do you want to install from sources the package which
needs compilation (yes/no/cancel)?”  Click yes
### Once packages are installed. Start code here.

library(BiocManager); packageVersion("BiocManager")

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

```
library(ShortRead); packageVersion("ShortRead")

library(ggplot2); packageVersion("ggplot2")

library(phyloseq); packageVersion("phyloseq")

library(Biostrings); packageVersion("Biostrings")

library(dplyr); packageVersion("dplyr")

library(easycsv)

library(devtools)

library(rBLAST); packageVersion("rBLAST")

library(tidyverse); packageVersion("tidyverse")

### Loading Data and Setting Filenames

### Before starting this pipeline, make sure you to set up the directory
appropriately. First, make a folder/directory to hold all subfolders
(e.g. “eDNA”). Within that folder make two subfolders called “data” and
“data-raw”. Within the “data-raw” folder make a folder called “fastqs”.
Put reference file “12S_ECO_refs_for_DADA2.fas” in the “data-raw” folder
and all fastq sample files in the “fastqs” subfolder. Set path to the
mother file wherever you will keep all your files (e.g. “eDNA”). After
running this code, the output files will be exported to the “data” folder
except the “blast.out” file, which will be exported to the “data-raw
folder”.

path =  choose_dir()

### blast results. This will create a file “blast.out” with identified
species. “blast.out” will be filtered to whatever you restrict your
sequence length to (default is 80-120bp, but you can modify it to your
needs by adjusting the min/max sequence length in this line from page 14:
### seqtab2 <- seqtab[,nchar(colnames(seqtab)) %in% seq(80,120)]
### A second file “blast.out.unfiltered” will include all sequences
regardless of length.

path1 = "data-raw/fastqs/"

blastout <- "data-raw/blast.out"

blastout2 <- "data-raw/blast.out.unflitered"



### output of dada2 sequences. This will create a fastq file “test.fna”
with all test sequences to be blasted.

tax_sequences <- "data/test.fna"
```

```
tax_sequences2 <- "data/test.fna.unfiltered"

vert_fasta <- "data-raw/12S_ECO_refs_for_DADA2.fas"

### Ensures your path contains all required files.

fns <- list.files(paste(path, path1, sep = ""))

fns

##  [1] "285AFS26jan2018_S56_L001_R1_001.fastq"
##  [2] "285AFS26jan2018_S56_L001_R2_001.fastq"
##  [3] "H2OAFS26jan2018-1_S16_L001_R1_001.fastq"
##  [4] "H2OAFS26jan2018-1_S16_L001_R2_001.fastq"
## [5] "H2OAFS26jan2018-2_S86_L001_R1_001.fastq"
## [6] "H2OAFS26jan2018-2_S86_L001_R2_001.fastq"




### Filtering and Trimming
### First we read in the file names for all the fastq files and do a
little string manipulation to get lists of the forward and reverse fastq
files in matched order:

fastqs <- fns[grepl(".fastq$", fns)]

fastqs <- sort(fastqs)

### Sort ensures Forward (R1) and Reverse (R2) reads correspond to each
other. Fastq files are paired end contain forward and reverse  reads.
### grepl searches for string or string vector, in this case any fastq
files ending in "_R1" or "_R2"

fnFs <- fastqs[grepl("_R1", fastqs)]

fnRs <- fastqs[grepl("_R2", fastqs)]

print(fnFs)

## [1] "285AFS26jan2018_S56_L001_R1_001.fastq"
## [2] "H2OAFS26jan2018-1_S16_L001_R1_001.fastq"
## [3] "H2OAFS26jan2018-2_S86_L001_R1_001.fastq"


print (fnRs)

## [1] "285AFS26jan2018_S56_L001_R2_001.fastq"
## [2] "H2OAFS26jan2018-1_S16_L001_R2_001.fastq"
## [3] "H2OAFS26jan2018-2_S86_L001_R2_001.fastq"


### Examine quality profiles of forward and reverse reads
```

It is always important to look at your data. There are many ways to do this, but here we use a visualization from the ShortRead package.

Visualize the quality profile of the forward reads:
it is essential for this code and subsequent code to follow the format as it is written in this script. If when copying to R/RStudio, breaks with light "|'s" appear, make sure to delete "|'s".

```
for(fnF in fnFs[1:1]) {
  print(fnF)
  qqF <- qa(paste0(path, path1, fnF))[["perCycle"]]$quality
  print(ShortRead:::.plotCycleQuality(qqF, main="Forward"))
}
```

```
## [1] "285AFS26jan2018_S56_L001_R1_001.fastq"
```

The forward reads are of good quality. In this case scores don't really drop below 30 until the end (30 arbitrary but stringent cutoff score). Green line is average quality score (most important). It is generally a good idea to trim the first 10 bases of Illumina sequences, as error rates are higher and less well-controlled at the start of Illumina sequencing. As quality dips towards end (around 150bp), it is also advisable to trim the very end as well (around 150bp). This ensures we will have accurate, high quality data. There is no suggestion from the quality profiles that any additional trimming is needed, so we will trim the first and last 10 nucleotides from the forward reads. Typically, you don't find much change between each file (assuming same protocol) so you don't need to check every file).

Visualize the quality profile of the reverse reads:

```
for(fnR in fnRs[1:1]) {
  qqR <- qa(paste0(path, path1, fnR))[["perCycle"]]$quality
  print(ShortRead:::.plotCycleQuality(qqR, main="Reverse"))
}
```

The reverse reads have significantly worse quality, especially towards the end of the reads, which is quite common in Illumina paired-end sequencing. This isn't too worrisome, DADA2 incorporates quality information into its error model so the algorithm is fairly robust to lower quality sequence, but some trimming as the average qualities crash is still a good idea. Here we will trim the first 10 nucleotides (as standard) and truncate at position 160 where the quality distribution crashes.

Perform filtering and trimming
The trimming parameters were decided by inspecting the quality profiles. The filtering parameters we'll use are standard: maxN=0 (DADA2 requires no Ns), truncQ=2 (quality score 2 in Illumina means "stop using

this read") and maxEE=2. The maxEE parameter sets the maximum number of "expected errors" allowed in a read. Setting a threshold on expected errors is a better filter than simply averaging quality scores. We use the fastqPairedFilter function to filter the forward and reverse reads together.

```
### Filter the forward and reverse reads:

filtFs <- paste0(path, sapply(strsplit(fnFs, "\\."), `[`, 1),
"_filt.fastq.gz")

filtRs <- paste0(path, sapply(strsplit(fnRs, "\\."), `[`, 1),
"_filt.fastq.gz")

for(i in seq_along(fnFs)) {
  fastqPairedFilter(paste0(path, path1, c(fnFs[i], fnRs[i])),
c(filtFs[i], filtRs[i]), maxN=0, maxEE=2, truncQ=2, trimLeft=c(18, 18),
truncLen=c(100,100), compress=TRUE, verbose=TRUE)
}
```

We now have trimmed and filtered fastq files. Check your directory to make sure new filtererd files "_filt.fastq.gz" are there before proceeding. The preceding filtering can be replaced by other filtering methods. However, in order for the later DADA2 mergePairs step to work, the filtered forward and reverse reads must be in matched order! The fastq files that come off the Illumina machine have this property, and fastqPairedFilter preserves it, but not all filtering tools do so.
first specifies forward and reverse reads (fnFs, fnRs) and where to put them (filtFs, fnRs). truncLen cuts forward and reverse reads at 100bp (first is forward, second is revrerse so if want to cut forward reads at 140 and reverse at 130 would write "truncLen=c(140, 130)". When considering cutting, need to understand length of amplicon and how much overlap you need. DADA2 needs 20bp overlap to successfully merge the reads (but can lower if need be). maxN species number of ambiguous base pairs that you can have in data. DADA2 currently can't handle ambiguous base pairs so by setting maxN=0 you get rid of all your reads that have ambiguous base pairs. maxEE is maximum estimated errors you would expect to see in a read. maxEE does a better job of estimating quality of read than just the quality score. Reads with more than number specified will be discarded. Typically keep as maxEE=2 so fewer expected errors and faster processing. With poor quality reads, where you can't sacrifice as many reads, you can increase maxEE. With higher quality you can drop maxEE to 1. truncQ truncates the read at the first instance of a Q score less than or equal to the specified value. Q score of 2 is terrible (~63% chance of base call being incorrect). This gets rid of worst reads that are unusable. truncQ=2 is good. compress=TRUE compresses fastq output and multithread=TRUE parallelizes action so reduces file size and speeds up process.

```
### Dereplication
```

```
### In the dereplication step, all reads with identical sequences are
combined into “unique sequences” with a corresponding abundance, i.e. the
number of reads with that unique sequence. Dereplication is a part of
most pipelines because it reduces lessens computation time by elminating
repeated comparisons of identical sequences.
### Dereplication in the DADA2 pipeline has a crucial difference: DADA2
retains a summary of the quality information associated with each unique
sequence. DADA2 constructs a “consensus” quality profile for each unique
sequence by averaging the positional qualities from the dereplicated
reads. These consensus quality profiles inform the error model of the
subsequent denoising step, significantly increasing DADA2’s accuracy.


### Dereplicate the filtered fastq files:
### lapply returns list object of same length as input list object.

derepFs <- lapply(filtFs, derepFastq, verbose=TRUE)

derepRs <- lapply(filtRs, derepFastq, verbose=TRUE)

### Name the derep-class objects by the sample names

sam_names <- sapply(strsplit(fnFs, "/"), tail, n=1)

sam_names <- sapply(strsplit(sam_names, "_"), `[`, 1)

names(derepFs) <- sam_names

names(derepRs) <- sam_names


### Inspect the derep-class object returned by derepFastq:

derepFs[[1]]

## derep-class: R object describing dereplicated sequencing reads
## $uniques: 206041 reads in 25311 unique sequences
##   Sequence lengths: min=82, median=82, max=82
## $quals: Quality matrix dimension:  25311 82
##   Consensus quality scores: min=12, median=34.5, max=40
## $map: Map from reads to unique sequences:  1 1 2 1 22939 ...

### Dereplicated sequences are stored in the $uniques integer vector,
which is named by the unique sequence and valued by the abundance of that
sequence. Consensus quality scores are stored in the $quals matrix: rows
correspond to unique sequences and columns to nucleotide position. The
$map vector maps the reads into the $uniques vector, and is used later
when we merge the forward and reverse reads.


### Sample Inference
### We are now ready to apply DADA2’s core sample inference algorithm to
the dereplicated sequences.
```

```
### But first a key consideration: DADA2 depends on a parametric error
model, and we do not know the error rates for this dataset. Fortunately,
DADA2 can jointly infer the error-rate parameters and the composition of
the sample, at the cost of additional computation time. This is done by
implementing an EM-like algorithm in which the error rates and the sample
are alternately estimated until convergence.
### To perform this joint inference with dada(...) we pass it the
selfConsist=TRUE flag, and specify the errorEstimationFunction =
loessErrfun (the current default option). As is common in optimization
problems we still must provide an initial guess at the error rates. For
this we take a previously estimated set of error rates (tperr1, included
with the package) and inflate them, as it is better to start with error
rates that are too high than too low.

### Perform joint sample inference and error rate estimation (takes a few
minutes):
### Creates an error model used by DADA2 algorithm downstream. Every
batch of sequencing has a different error rate. In order to get starting
point, takes most abundant sequence and assumes that's the only sequence
that is actually true and all other sequences are caused by errors. Then
alternates error rate estimation and inferring sample composition until
it arrives at solution that makes sense for both parameters. Takes more
steps in reverse reads as they are typically worse.

dadaFs <- dada(derepFs, err=inflateErr(tperr1,3),
errorEstimationFunction=loessErrfun, selfConsist = TRUE)

## Sample 1 - 206041 reads in 25311 unique sequences.
## Sample 2 - 10032 reads in 8711 unique sequences.
## Sample 3 - 36381 reads in 28176 unique sequences.
##          selfConsist step 2
##          selfConsist step 3
##          selfConsist step 4
##          selfConsist step 5
##          selfConsist step 6
##
##
## Convergence after  6  rounds.

dadaRs <- dada(derepRs, err=inflateErr(tperr1,3),
errorEstimationFunction=loessErrfun, selfConsist = TRUE)

## Sample 1 - 206041 reads in 37278 unique sequences.
## Sample 2 - 10032 reads in 9869 unique sequences.
## Sample 3 - 36381 reads in 30597 unique sequences.
##          selfConsist step 2
##          selfConsist step 3
##          selfConsist step 4
##          selfConsist step 5
##
##
## Convergence after  5  rounds.

### Inspecting the dada-class object returned by dada:
```

```
dadaFs[[1]]

## dada-class: object describing DADA2 denoising results
## 118 sample sequences were inferred from 25311 input unique sequences.
## Key parameters: OMEGA_A = 1e-40, BAND_SIZE = 16, USE_QUALS = TRUE
# The dada algorithm inferred 130 real sequences out of 1860 input unique
sequences in the first sample. There is much more to the dada-class
return object than this (see help("dada-class") for some info), including
multiple diagnostics about the quality of the inference, but that is a
subject for another tutorial. Let's do one check on the quality of the
error-rate estimation though before continuing though.



### Visualize estimated error rates:

plotErrors(dadaFs[[1]], "A", nominalQ=TRUE)


### Here we only plotted the error rates from A. The points are the
observed error rates for each consensus quality score. The black line is
the estimated error rate after convergence. The red line is the error
rate expected under the nominal definition of the Q-value. Plots
frequency of each possible base transition (probability an A is actually
read as a C). As quality score increases the expected error frequencies
decrease.

### Everything looks reasonable here, and we can proceed with confidence.

### Identify chimeras
### The dada() algorithm removes substition and indel errors, but it does
not remove chimeras. If chimeras are present in the sequenced sample,
they will be found and reported. Therefore, we now identify chimeras.

### The accuracy of the sequences after the dada-denoising step makes
identifying chimeras easier than it is when dealing with fuzzy OTUs. The
DADA2 method to do this is isBimeraDenovo, which does de-novo
identification of bimeras (two-parent chimeras) on a sample-by-sample
basis.




### Identify chimeric sequences:

bimFs <- sapply(dadaFs, isBimeraDenovo, verbose=TRUE)


bimRs <- sapply(dadaRs, isBimeraDenovo, verbose=TRUE)
```

```
print(unname(sapply(bimFs, mean)), digits=2)

## [1] 0.093 0.034 0.200

print(unname(sapply(bimRs, mean)), digits=2)

## [1] 0.42 0.00 0.00

### The fraction of chimeras varies between samples, but can be
substantial. Here chimeras make up as much as 27% of the inferred
sequences in the 9th sample. We will remove these chimeras in the next
step after merging the forward and reverse reads.



### Merge paired reads

### We now merge the denoised forward and reverse reads together. Note
that in the DADA2 pipeline merging is perfomed after denoising the
forward and reverse reads independently. The core function here is
mergePairs. Remember that mergePairs depends on the forward and reverse
reads being in matching order at the time they were dereplicated!

### Merge the denoised forward and reverse reads:

mergers <- mapply(mergePairs, dadaFs, derepFs, dadaRs, derepRs,
SIMPLIFY=FALSE)

head(mergers[[1]])


### We now have a data.frame for each sample with the merged $sequence,
its $abundance, and the indices of the merged $forward and $reverse
denoised sequences. Paired reads that did not exactly overlap were
removed by mergePairs. We now remove the chimeras we identified
previously in the forward and reverse denoised sequences:
### ** Remove chimeras **:

mergers.nochim <- mapply(function(mm, bF, bR) mm[!bF[mm$forward] &
!bR[mm$reverse],], mergers, bimFs, bimRs, SIMPLIFY=FALSE)


### Constructing the sequence table
### Sample inference is complete, and we can now construct the “sequence
table” analogous to the “OTU table” produced by OTU methods. We drop the
Mock community at this point.
## seqtab <- makeSequenceTable(mergers[names(mergers) != "Mock"])

seqtab <- makeSequenceTable(mergers.nochim)

### The sequences being tabled vary in length.

seqtab2 <- seqtab[,nchar(colnames(seqtab)) %in% seq(80,120)]
```

```
table(nchar(colnames(seqtab2)))

##
## 82 86 87 90 92 97104105106107108111127133142143
## 1 1 2 1 1 4 2 7 12 20 3 1 3 1 1 1

length(unique(substr(colnames(seqtab2), 1, 100)))

## [1] 60

dim(seqtab2)

##[1] 361



### Bonus: Further analysis in phyloseq
### The DADA2 pipeline produced a sequence table which is appropriate for
further analysis in phyloseq. We don't have all the information we might
want yet, for instance we have not yet assigned taxonomies, but we can
import the data we have into phyloseq and perform some basic analyses.
We'll include the small amount of metadata we have - the samples are
named by the gender (G), mouse subject number (X) and the day post-
weaning (Y) it was sampled (eg. GXDY).

### Import into phyloseq:
### Make "otu_table"

seqs <- colnames(seqtab2)


otab <- otu_table(seqtab2, taxa_are_rows=FALSE)


colnames(otab) <- paste0("Seq_", seq(ncol(otab)))


### Get Taxonomy Based on Blasting against a known reference
### Write fastas to test file

writeFasta <- function(seqs, output) {
  seqsout <- mapply( function(idx, sequence)
paste0(">Seq_",idx,"\n",sequence,"\n"), seq(length(seqs)), seqs)
  write(paste0(seqsout), file = output, sep = "")
}

seqs_for_blast <- DNAStringSet(seqs)


names(seqs_for_blast) <- sapply(seq(length(seqs)),function(x)
{paste0("Seq_",x)})

writeFasta(seqs, paste(path, tax_sequences, sep = ""))
```

```
transotab = t(otab)

write.table(transotab, paste0(path, "data/otutable.txt"))

write.csv(transotab, paste0(path, "data/otutable.csv"))


### Here you will ask R to run the rBLAST package on your operating
system, rather than through R by using the system command.

system(paste("makeblastdb -dbtype nucl -in", paste0(path,vert_fasta)))

### any additional column headers can be added within the single
quotations. Additional column headers can be found on
https://www.metagenomics.wiki/tools/blast/blastn-output-format-6. Order
is important as when naming the columns using colnames function, it must
be in same order as below. The 6 is the blastout format - makes it tab
deliminated format, while remaining criteria in single quotations are
columns.

system(paste("blastn -query", paste0(path, tax_sequences),  "-db",
paste0(path, vert_fasta), "-outfmt '6 qseq qseqid sseqid pident length
mismatch gapopen qstart qend sstart send evalue bitscore' -
max_target_seqs 1 -out", paste0(path, blastout)))

### Can rename columns but must be in same order as above. In this
example, qseq has been names qid and qse


blasttable <- read.table(paste0(path, blastout))


blasttable


colnames(blasttable) <- c("qseq", "qid", "sid", "pident", "length",
"mismatch", "gapopen", "qstart", "qend", "sstart", "send", "evalue",
"bitscore")



paste("blastn -query", paste0(path, tax_sequences),  "-db", paste0(path,
vert_fasta), "-outfmt 6 -max_target_seqs 1 -out", paste0(path, blastout))



# paste("blastn -query", tax_sequences,  "-db", vert_fasta, "-outfmt '6
qseq qseqid sseqid pident length mismatch gapopen qstart qend sstart send
evalue bitscore' -max_target_seqs 1 -out", blastout))
```

```
## [1] "blastn -query  data/test.fna -db data-raw
/12S_ECO_refs_for_DADA2.fas -outfmt 6 -max_target_seqs 1 -out  data-
raw/blast.out"


### load the blast table back in and use it to make a taxonomy table

read.table(paste0(path, blastout))




### pident  == 100 means that the table will exclude any sequences that
don't match 100%

blastresults <- blasttable %>%
  filter(pident == 100) %>%
  select(qid, sid) %>%
  data.frame()


taxtab <- data.frame(seqs = colnames(otab), stringsAsFactors = FALSE) %>%
  left_join(blastresults, by=c('seqs'='qid'))



taxtab$taxa <- taxtab$sid


rownames(taxtab) <- taxtab$seqs



taxtab <- taxtab %>%
  select(-seqs, -sid) %>%
  as.matrix() %>%
  tax_table


ps <- phyloseq(otab, taxtab)


ps

## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 185 taxa and 3 samples ]
## tax_table()   Taxonomy Table:    [ 185 taxa by 1 taxonomic ranks ]

top10 <- names(sort(taxa_sums(ps), decreasing=TRUE))[1:10]

ps.top10 <- transform_sample_counts(ps, function(OTU) OTU/sum(OTU))

ps.top10 <- prune_taxa(top10, ps.top10)
```

```
plot_bar(ps.top10, fill="taxa")



tax_table1<-"data/taxtable.txt"

write.table(taxtab, file=paste0(path,tax_table1))


paste("blastn -query", tax_sequences,  "-db", vert_fasta, "-outfmt '6
qseq qseqid sseqid pident length mismatch gapopen qstart qend sstart send
evalue bitscore' -max_target_seqs 1 -out", blastout)

### To make taxonomy table a csv so it can be automatically delimited in
excel

tax_tablecsv<-"data/taxtable.csv"

write.csv(taxtab, file=paste0(path,tax_tablecsv))

### To make blasttable a csv so it can be automatically delimited in
excel

blasttable <- read.table(paste0(path, blastout))

colnames(blasttable) <- c("qseq", "qid", "sid", "pident", "length",
"mismatch", "gapopen", "qstart", "qend", "sstart", "send", "evalue",
"bitscore")

blast_out<-"data-raw/blast.out.csv"

write.csv(blasttable,file=paste0(path,blast_out))









table(nchar(colnames(seqtab)))

##
## 82 86 87 90 92 97104105106107108111127133142143
## 1 1 2 1 1 4 2 7 12 20 3 1 3 1 1 1

length(unique(substr(colnames(seqtab), 1, 100)))

## [1] 60
```

```
dim(seqtab)

##[1] 361



### Bonus: Further analysis in phyloseq
### The DADA2 pipeline produced a sequence table which is appropriate for
further analysis in phyloseq. We don't have all the information we might
want yet, for instance we have not yet assigned taxonomies, but we can
import the data we have into phyloseq and perform some basic analyses.
We'll include the small amount of metadata we have - the samples are
named by the gender (G), mouse subject number (X) and the day post-
weaning (Y) it was sampled (eg. GXDY).

### Import into phyloseq:
### Make "otu_table"

seqs2 <- colnames(seqtab)


otab2 <- otu_table(seqtab, taxa_are_rows=FALSE)


colnames(otab2) <- paste0("Seq_", seq(ncol(otab2)))


### Get Taxonomy Based on Blasting against a known reference
### Write fastas to test file

writeFasta2 <- function(seqs2, output) {
  seqsout2 <- mapply( function(idx, sequence)
paste0(">Seq_",idx,"\n",sequence,"\n"), seq(length(seqs2)), seqs2)
  write(paste0(seqsout), file = output, sep = "")
}

seqs_for_blast2 <- DNAStringSet(seqs2)


names(seqs_for_blast2) <- sapply(seq(length(seqs2)),function(x)
{paste0("Seq_",x)})

writeFasta(seqs2, paste(path, tax_sequences2, sep = ""))







transotab2 = t(otab2)
```

```
write.table(transotab2, paste0(path, "data/otutable_unfiltered.txt"))

write.csv(transotab2, paste0(path, "data/otutable_unflitered.csv"))


### Here you will ask R to run the rBLAST package on your operating
system, rather than through R by using the system command.

### any additional column headers can be added within the single
quotations. Additional column headers can be found on
https://www.metagenomics.wiki/tools/blast/blastn-output-format-6. Order
is important as when naming the columns using colnames function, it must
be in same order as below. The 6 is the blastout format - makes it tab
deliminated format, while remaining criteria in single quotations are
columns.

system(paste("blastn -query", paste0(path, tax_sequences2),  "-db",
paste0(path, vert_fasta), "-outfmt '6 qseq qseqid sseqid pident length
mismatch gapopen qstart qend sstart send evalue bitscore' -
max_target_seqs 1 -out", paste0(path, blastout2)))

### Can rename columns but must be in same order as above. In this
example, qseq has been names qid and qse


blasttable2 <- read.table(paste0(path, blastout2))


blasttable2


colnames(blasttable2) <- c("qseq", "qid", "sid", "pident", "length",
"mismatch", "gapopen", "qstart", "qend", "sstart", "send", "evalue",
"bitscore")



paste("blastn -query", paste0(path, tax_sequences2),  "-db", paste0(path,
vert_fasta), "-outfmt 6 -max_target_seqs 1 -out", paste0(path,
blastout2))



# paste("blastn -query", tax_sequences2,  "-db", vert_fasta, "-outfmt '6
qseq qseqid sseqid pident length mismatch gapopen qstart qend sstart send
evalue bitscore' -max_target_seqs 1 -out", blastout2))


## [1] "blastn -query  data/test.fna.unfiltered -db data-raw
/12S_ECO_refs_for_DADA2.fas -outfmt 6 -max_target_seqs 1 -out  data-
raw/blast.out2"


### load the blast table back in and use it to make a taxonomy table
```

```
read.table(paste0(path, blastout2))



### pident  == 100 means that the table will exclude any sequences that
don't match 100%

blastresults2 <- blasttable2 %>%
  filter(pident == 100) %>%
  select(qid, sid) %>%
  data.frame()


taxtab2 <- data.frame(seqs2 = colnames(otab2), stringsAsFactors = FALSE)
%>%
  left_join(blastresults2, by=c('seqs2'='qid'))



taxtab2$taxa <- taxtab2$sid


rownames(taxtab2) <- taxtab2$seqs2


taxtab2 <- taxtab2 %>%
  select(-seqs2, -sid) %>%
  as.matrix() %>%
  tax_table


ps2 <- phyloseq(otab2, taxtab2)


ps2

## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 185 taxa and 3 samples ]
## tax_table()   Taxonomy Table:    [ 185 taxa by 1 taxonomic ranks ]



tax_table2<-"data/taxtable_unfiltered.txt"

write.table(taxtab2, file=paste0(path,tax_table2))


paste("blastn -query", tax_sequences2,  "-db", vert_fasta, "-outfmt '6
qseq qseqid sseqid pident length mismatch gapopen qstart qend sstart send
evalue bitscore' -max_target_seqs 1 -out", blastout2)
```

```
### To make taxonomy table a csv so it can be automatically delimited in
excel

tax_tablecsv2<-"data/taxtable_unfiltered.csv"

write.csv(taxtab2, file=paste0(path,tax_tablecsv2))

### To make blasttable a csv so it can be automatically delimited in
excel

blasttable2 <- read.table(paste0(path, blastout2))

colnames(blasttable2) <- c("qseq", "qid", "sid", "pident", "length",
"mismatch", "gapopen", "qstart", "qend", "sstart", "send", "evalue",
"bitscore")

blast_out2 <-"data-raw/blast.out.unfiltered.csv"

write.csv(blasttable2,file=paste0(path,blast_out2))
```